

Solving the Consolidation Problem

Now we mean business

Knowing Your Enemy

Now that we know better, let's look again at the consolidation problem:

$$\begin{aligned} & \arg \min_x \|z\|_1 \\ & \text{subject to: } Vx = v^*, Ex = e^* \\ & \quad x \leq Mz \\ & \quad x \geq 0, z \in \{0, 1\}^n \end{aligned}$$

Where do you think most of the complexity will stem from?

Knowing Your Enemy

Now that we know better, let's look again at the consolidation problem:

$$\begin{aligned} & \arg \min_x \|z\|_1 \\ & \text{subject to: } Vx = v^*, Ex = e^* \\ & \quad x \leq Mz \\ & \quad x \geq 0, z \in \{0, 1\}^n \end{aligned}$$

Where do you think most of the complexity will stem from?

- We need to branch only on the **integer variables**
- So, their number will likely have an impact on complexity

Since we focus on the current Path formulation solution, **they won't be many**

Knowing Your Enemy

Now that we know better, let's look again at the consolidation problem:

$$\begin{aligned} & \arg \min_x \|z\|_1 \\ & \text{subject to: } Vx = v^*, Ex = e^* \\ & \quad x \leq Mz \\ & \quad x \geq 0, z \in \{0, 1\}^n \end{aligned}$$

What could you say of the impact of using big-Ms?

Knowing Your Enemy

Now that we know better, let's look again at the consolidation problem:

$$\begin{aligned} & \arg \min_x \|z\|_1 \\ & \text{subject to: } Vx = v^*, Ex = e^* \\ & \quad x \leq Mz \\ & \quad x \geq 0, z \in \{0, 1\}^n \end{aligned}$$

What could you say of the impact of using big-Ms?

- Logically, they work just fine
- In practice, they can lead to **poor bounds** in the LP relaxation

Ideally, they should be **avoided**. Failing that, use an **M** as small as possible

The Solution Code

The code for solving the problem is in the `consolidate_paths` function

The function parameter look similar to those of `solve_path_selection_full`

```
def consolidate_paths(  
    tug : ig.Graph,  
    paths : list,  
    node_counts : dict,  
    arc_counts : dict,  
    tlim : int = None):
```

However, they are meant to be **used differently**:

- `paths` should contain those selected by the Path formulation
- `node_counts` should contain the counts from the Path formulation solution
- ...And the same goes for `arc_counts`

`tlim` is a time limit: always use one when dealing with NP-hard problems

The Solution Code

Let's see some relevant code snippets:

We use the CBC solver, via the Google Or-Tools Wrapper

```
slv = pywraplp.Solver.CreateSolver('CBC')
```

Variables are built using the solver object and stored in lists:

```
x = [slv.NumVar(0, inf, f'x_{j}') for j in range(npaths)]  
z = [slv.IntVar(0, 1, f'z_{j}') for j in range(npaths)]
```

For the big-M constraints ($x \leq Mz$) we use the largest node count

```
M = max(v for v in node_counts.values())  
for j in range(npaths):  
    slv.Add(x[j] <= M * z[j])
```

- There no need for a path to use a flow larger than that

The Solution Code

Let's see some relevant code snippets:

Here's the code for the "count matching" constraints, i.e. $\sum x = v^*$ and $\sum Ex = e^*$:

```
for n, p in paths_by_node.items():
    slv.Add(sum(x[j] for j in p) == node_counts[n])
for a, p in paths_by_arc.items():
    slv.Add(sum(x[j] for j in p) == arc_counts[a])
```

- We rely on a previous step where we grouped path by used node/arc

Here's how we define the objective and optimization direction:

```
slv.Minimize(sum(z[j] for j in range(npaths)))
```

...And here how to set a time limit:

```
if tlim is not None: slv.SetTimeLimit(tlim)
```


The Solution Code

Let's see some relevant code snippets:

We trigger the solution process with the `solve` method:

```
status = slv.Solve()
```

The method returns an integer `status code`, that should always be checked:

```
if status in (slv.OPTIMAL, slv.FEASIBLE):  
    # Extract the paths in the solution  
    ...  
    # Return the solution  
    if status == slv.OPTIMAL: return sol_flows, sol_paths, True  
    else: return sol_flows, sol_paths, False  
else:  
    return None, None, False
```

- If we find a solution we return it
- If we prove optimality within the time limit, we tell it with a flag

Solving the Problem

We can finally solve the consolidation problem for real:

```
In [3]: node_counts_r, arc_counts_r = util.get_counts(tug, rflows, rpaths)
        cflows, cpaths, cflag = util.consolidate_paths(tug, rpaths, node_counts_r, arc_counts_r)
        print('FLOW: PATH')
        util.print_solution(tug, cflows, cpaths, sort='descending')
        print(f'Optimal: {cflag}')
```

```
FLOW: PATH
8.17: 2,3 > 3,3
5.47: 0,2 > 1,2 > 2,2 > 3,2
4.89: 0,1 > 1,1 > 2,0 > 3,0
3.74: 3,3
3.32: 1,0 > 2,0 > 3,2
Optimal: True
```

In our case, the consolidated paths match the ground truth perfectly!

```
In [4]: print('FLOW: PATH')
        util.print_ground_truth(flows, paths, sort='descending')
```

```
FLOW: PATH
8.17: 2,3 > 3,3
5.47: 0,2 > 1,2 > 2,2 > 3,2
4.89: 0,1 > 1,1 > 2,0 > 3,0
```