Office Workspace Reallocation Project

Federal Reserve Bank of Richmond, VA

Bryce Bowles

Faitha Schrader

SCMA 645 Management Science
Virginia Commonwealth University
Spring 2020

The IT functional groups hosted within the Federal Reserve Bank of Richmond reorganize approximately every six months and have asked for help with allocating their office spaces. We, Bryce Bowles and Faitha Schrader, students of Virginia Commonwealth University's School of Business, Masters of Decision Analytics, are taking Management Science with Dr. Paul Brooks. The task is for us to create an optimization model using the data provided by Vice president of IT, Christine Holzem. Once we were given the initial assignment information, we met with Christine to clarify some of our assumptions and get additional information. We created a sample model for submission within the first two weeks. After we received the data set for current workers, we had another two weeks to adjust our base model to incorporate the new data and write the final report.

We created an optimization model, reallocating employees, to minimize the number of times an employee will move workspaces and the distance between team members. We used an "assignment" network optimization problem approach with binary variables and logical constraints. This model can easily be adapted to any floor by updating the information in the spreadsheet. We did this due to the ever-changing environment at the Federal Reserve Banks, whose teams can operate in 9 different states and change projects frequently, creating the need to generate a continuous model for future use.

The optimal solution for the 20 workers requires 11 moves from their current locations and the total distance between members of all teams is 1,230ft. This is broken down into: Team A 110 ft., Team B 550 ft., and Team C 570 ft. It also ensures that the rank of the office space is equivalent with the rank of the worker. We recommend using this optimization model for each floor, whereas you would only need to change the data in the spread sheet.

Problem:

The Federal Reserve needs to optimize seating locations of 1700 workers on 17 floors, so approximately 100 workers per floor with half teleworking from home. Office spaces are assigned based on rank, and team members need to be in close proximity of one another. The goal is to minimize moves of current workers and minimize distance between team members. A typical floor plan is 110ft x 110ft with four sides (A,B,C,D) and consist of 4 corner offices, 4 3-window offices, 8 2-window offices, 24 exterior cubicles, 24 interior cubicles, 8 interior offices, and two conference rooms (see appendix1, Floorplan). For information on the full assignment, (see appendix 4).

We started with a smaller base model of seven workers having seven offices and decided an "assignment" optimization problem to reassign workers to spaces would be most effective. With this, we began making assumptions for moves and distances (see appendix 2, moves and 3, distance). This allowed us to work out a model that easily adapted to the complete floor plan (see appendix1, Floorplan). The distance spreadsheet is based off the floor plan of how the office spaces are aligned on each side of the building. The moves spreadsheet assigns current offices to the workers but can be updated to address the current office layout. The worker data spreadsheet has current worker data; however, it can be adjusted with actual worker name and information to provide updated results (see appendix 4, worker).

Data:

Let:

W = {1 - 20} Workers

S = {ACO, A2W1, A3W, AEC, AIC, A2W2, , B2W1, B3W, BEC, BIC, BIO1, BIO2, B2W2} For the spaces available we assigned letter codes, for instance: (ACO = A side corner office or BEC = B side exterior cubicle)

V = W U S= { 1,2,3……. ACO, A2W1, A2W2}

A = W X S = {(1, ACO), (1, A2W1),……}

D = $S \cup S$

m_{ij} = move of $worker_i$ to $space_i$ $i, j \in A$

d_{jl} = distance from $space_j$ to $space_l$ $j, l \in D$

b_i = { 1 if $i \in S$ and -1 if $i \in T$

rank_worker = rank of worker

rank_space = rank allowed in space

team_worker = team assignment of worker

capacity = 7 (capacity of office space)

Objective in words

Assign workers to office spaces so that distance between workers on a team are minimized and the number of workers moving to a new space is minimized, subject to the following constraints:

- Capacity of space is equal to 1
- Worker rank is equal to space rank
- Moves are minimized
- Distance between team members is minimized

Algebraic Formulation:

Decision Variables:

Let:

x_{ij} = 1 if worker$_i$ is assigned to space$_j$, $i, j \in A$

y_{ijkl} = 1 if team member$_i$ is assigned to space$_j$ and team member$_k$ is assigned to space$_l$, $(i, j) \in A$, $(k, l) \in A$

Objective:

$$\min \sum_{i,j \in A} m_{ij} x_{ij} \quad + \quad \sum_{i \in W} \sum_{j \in S} \sum_{k \in W} \sum_{l \in S} d_{jl} y_{ijkl} \qquad \text{(Moves\_distance)}$$

Constraints:

$$\sum_{j \in V:(j,i) \in A} x_{ji} \; - \sum_{j \in V:(i.j) \in A} x_{ij} \; = \; b_i \, , i \in W \;\; (Assign\ Worker)$$

$$\sum_{j \in V:(j,i) \in A} x_{ji} \; - \sum_{j \in V:(i.j) \in A} x_{ij} \; \geq \; b_i \, , i \in W \; (Space\ Limit)$$

$$\text{if team\_}worker_i = \text{team\_}worker_i \colon x_{ij} + x_{kl} \; \leq y_{ijkl} \; + 1, (\, ij), (kl) \in A$$

$$else\colon \;\; y_{ijkl} \; = 0 \quad (Flipon\ Distance)$$

$$\text{if rank\_}worker_i \neq \text{rank\_}space_j \colon \quad x_{ij} \; = 0, (\, ij) \in A \quad (Rank\ Constraint)$$

Method:

Our optimization model uses an objective function that calculates the total moves based off the spreadsheet "Moves"(see appendix 2) for any worker that has to relocate from their current space and added the total distance based off the spreadsheet "Distance"(see appendix 3) for the distance from $space_j$ to $space_l$. We then added the constraint to ensure workers are assigned a space and workspace capacity for cubicles is 7. We only did the cubicle capacity because

workspace rank assignment would limit the office assignment on each team. Next, we have a “flip on” constraint for distance that will only add the distance if workers are assigned to the same team. Lastly, we provided a constraint that will only assign spaces that have a rank equivalent to the workers rank. For instance, A side corner office is only assigned to a worker with rank 1. Once setting up the model with Python, we used CoCalc/Anaconda’s Jupyter Lab to run the model and obtained a solution with “glpk” solver.

Implementation:

See attached Jupyter Notebook file (appendix 6, Python)

Results:

The following table shows the eleven workers required to move (in blue) and the total distance between team members. Workers 1, 2, 5, 7, 8, 9, 10, 11, 16, 18, and 20 will acquire new workspace locations. The distance column is the sum of the distances between that individual and the other team members.

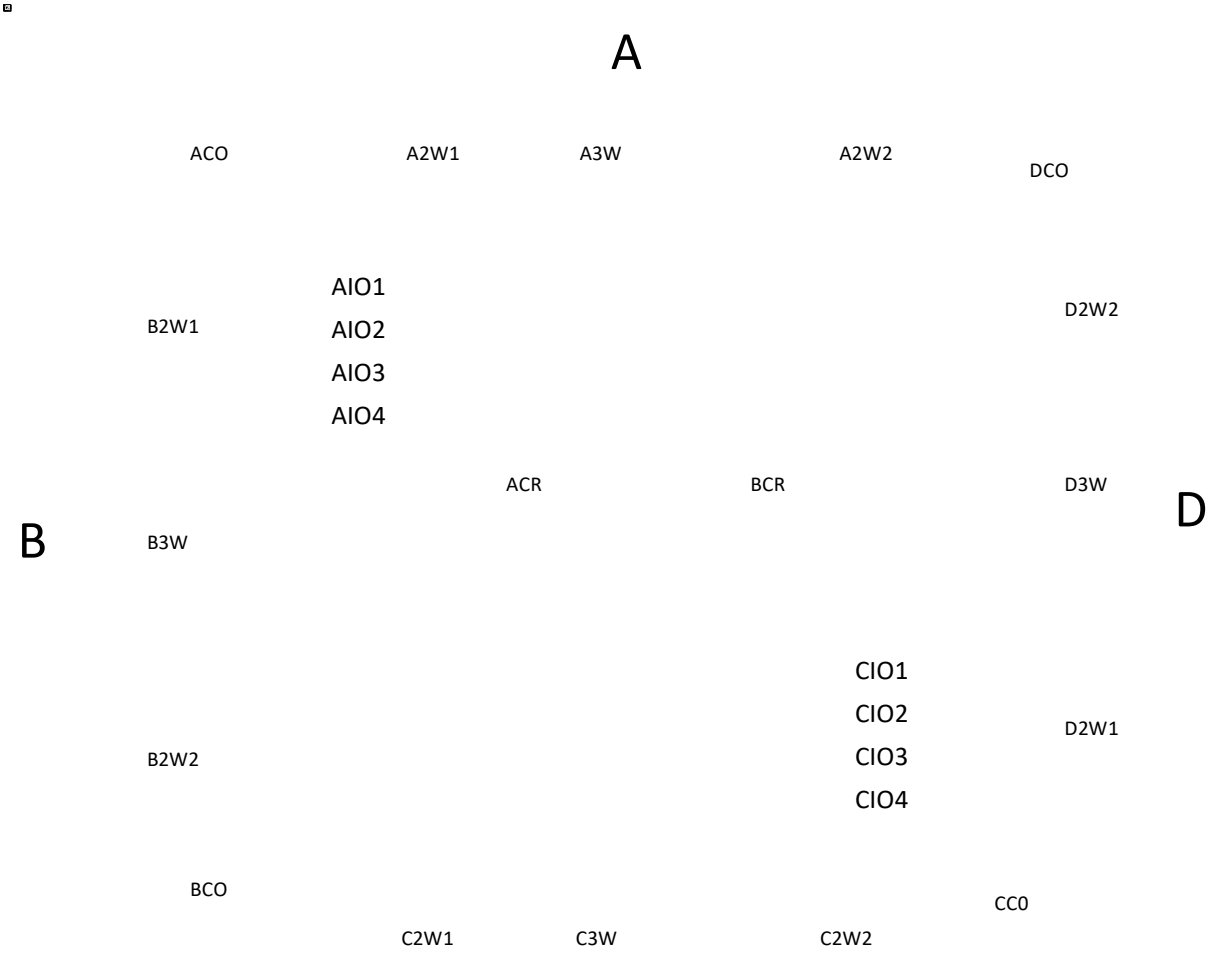
Worker	Rank	Team	Space Rank	Current Location	New Location	Distance	Average Distance	Total Team Distance
1	1	1	1	A2W1	B3W	35	27.50	110
2	3	1	3	AEC	BEC	20		
3	3	1	3	BEC	BEC	20		
4	4	1	4	BIC	BIC	35		
5	2	2	2	BIO2	BIO1	215	91.67	550
6	3	2	3	BEC	BEC	65		
7	3	2	4	BIC	BEC	65		
8	4	2	4	AEC	BIC	70		
9	3	2	3	AIC	BEC	65		
10	4	2	4	AIC	BIC	70		
11	1	3	1	B3W	A3W	105	57	570
12	3	3	3	AEC	AEC	40		
13	4	3	4	AIC	AIC	75		
14	3	3	3	AEC	AEC	40		
15	4	3	4	AIC	AIC	75		
16	3	3	3	BEC	AEC	40		
17	3	3	3	AEC	AEC	40		
18	3	3	3	BEC	AEC	40		
19	4	3	4	AIC	AIC	75		
20	3	3	3	BEC	AEC	40		
					11 Moves			1230= Total ft

Conclusion and Recommendations:

While moving a little over half of the workers and a total distance of 1,230 feet may sound like a lot, you will see a great improvement in productivity and efficiency between team members. You could make an assumption that Team 1 will benefit the most out of this move as they attained the least average distance between team members while also having a relatively medium move rate (of 50%) and Team 2 will benefit the least considering 5/6 workers had to move workspaces while obtaining the greatest average distance (91.67ft) between team members.

The strength of this model is the ability to adjust it's data set to accommodate a greater volume of attributes and values. When implementing new data and increasing the scale, we recommend starting with one floor at a time; creating constraints specific for a floor and then adding complexities if teams need to be divided onto more than one floor. Something to consider are scenarios with exceptions, for example, where a worker is permitted to sit in a workspace outside of their rank. If the model recognizes a worker as currently sitting in a workspace rank higher than allowed, then it will demote them when running the model. Always spot check and troubleshoot for exceptions in the results. In this data set, worker 5 stands out as exceptionally far away from their teammates, therefore, skewing the average distance for Team 2. For this individual, we would suggest you consider approaching facilities management to see what workspace they could possibly add to make room for this individual to be closer. After all, there may be a possibility we see an increase in people working from home in the future.

Appendix 1: Floorplan



Appendix 2 – Moves

Worker	ACO	A2W1	A3W	AEC	AIC	A2W2	BCO	B2W1	B3W	BEC	BIC	BIO1	BIO2	B2W2
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	0	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	0	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	0	1
6	1	1	1	1	1	1	1	1	1	0	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	0	1	1	1
8	1	1	1	0	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1	1	1	1	1	1	1	1
10	1	1	1	1	0	1	1	1	1	1	1	1	1	1
11	1	1	0	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	0	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	0	1	1	1	1	1	1	1	1	1
14	1	1	1	0	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	0	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	0	1	1	1	1
17	1	1	1	0	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	0	1	1	1	1
19	1	1	1	1	0	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	0	1	1	1	1

Appendix 3: Distance

Spaces	ACO	A2W1	A3W	AEC	AIC	A2W2	BCO	B2W1	B3W	BEC	BIC	BIO1	BIO2	B2W2
ACO	0	10	15	20	25	90	95	100	105	110	115	150	155	190
A2W1	10	0	10	15	20	85	90	95	100	105	110	145	150	185
A3W	15	10	0	10	15	80	85	90	95	100	105	140	145	180
AEC	20	15	10	0	10	75	80	85	90	95	100	135	140	175
AIC	25	20	15	10	0	70	75	80	85	90	95	130	135	170
A2W2	90	85	80	75	70	0	10	15	20	25	30	65	70	105
BCO	95	90	85	80	75	10	0	10	15	20	25	60	65	100
B2W1	100	95	90	85	80	15	10	0	10	15	20	55	60	95
B3W	105	100	95	90	85	20	15	10	0	10	15	50	55	90
BEC	110	105	100	95	90	25	20	15	10	0	10	45	50	85
BIC	115	110	105	100	95	30	25	20	15	10	0	40	45	80
BIO1	150	145	140	135	130	65	60	55	50	45	40	0	10	45
BIO2	155	150	145	140	135	70	65	60	55	50	45	10	0	40
B2W2	190	185	180	175	170	105	100	95	90	85	80	45	40	0

Appendix 4: Worker

Worker	Rank	Team	Current Location
1	1	1	A2W1
2	3	1	AEC
3	3	1	BEC
4	4	1	BIC
5	2	2	BIO2
6	3	2	BEC
7	3	2	BIC
8	4	2	AEC
9	3	2	AIC
10	4	2	AIC
11	1	3	B3W
12	3	3	AEC
13	4	3	AIC
14	3	3	AEC
15	4	3	AIC
16	3	3	BEC
17	3	3	AEC
18	3	3	BEC
19	4	3	AIC
20	3	3	BEC

Appendix 6: Python


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Federal_Reserve_Bank_Workspace_Final_Report\n",
    "Virginia Commonwealth University\n",
    "\n",
    "Faitha Schrader and Bryce Bowles\n",
    "\n",
    "Spring 2020\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Read data."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "{1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 19: 1, 20: 1, 'ACO': -1, 'A2W1': -1, 'A3W': -1, 'AEC': -7, 'AIC': -7, 'A2W2': -6, 'BCO': -1, 'B2W1': -1, 'B3W': -1, 'BEC': -6, 'BIC': -6, 'BIO1': -1, 'BIO2': -1, 'B2W2': -1}\n"
     ]
    }
   ],
   "source": [
    "import pandas as pd\n",
    "\n",
    "move_data = pd.read_excel(\"Fed_Res_w20o14.xlsx\",\n",
    "                          sheet_name=\"Moves\",\n",
    "                             index_col=0)\n",
    "S = list(move_data)\n",
    "W = list(move_data.index)\n",
    "m = move_data.stack().to_dict()\n",
    "A = list(m.keys())\n",
```

```
   "V = W+S\n",
   "\n",
   "distance_data = pd.read_excel(\"Fed_Res_w20o14.xlsx\",\n",
   "                      sheet_name=\"Distance\",\n",
   "                        index_col=0)\n",
   "d = distance_data.stack().to_dict()\n",
   "D = list(d.keys())\n",
   "\n",
   "worker_data = pd.read_excel(\"Fed_Res_w20o14.xlsx\",\n",
   "                      sheet_name=\"Worker\",\n",
   "                        index_col=0)\n",
   "rank_worker = dict(zip(move_data.index, worker_data.Rank))\n",
   "team_worker = dict(zip(move_data.index, worker_data.Team))\n",
   "space_data = pd.read_excel(\"Fed_Res_w20o14.xlsx\",\n",
   "                      sheet_name=\"Space\",\n",
   "                        index_col=0)\n",
   "rank_space = dict(zip(move_data, space_data.Rank))\n",
   "\n",
   "capacity = {'ACO':-1, 'A2W1':-1, 'A3W': -1, 'AEC': -7, 'AIC': -7, 'A2W2': -6, 'BCO': -1, 'B2W1':-1,'B3W':-
1,'BEC':-6,'BIC':-6,'BIO1':-1,'BIO2':-1,'B2W2':-1}\n",
   "\n",
   "b = dict(zip(W, [1]*len(W)))\n",
   "b.update(capacity)\n",
   "\n",
   "print(b)\n",
   "\n",
   "def mybounds(model, i,j):\n",
   "    return (0.0, 1.0)\n",
   "def mybounds2(model, i,j,k,l):\n",
   "    return (0.0, 1.0)\n"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "Import Pyomo and define model."
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 2,
  "metadata": {},
  "outputs": [],
  "source": [
   "from pyomo.environ import *\n",
   "model = ConcreteModel()"
  ]
```

```
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "Let $x_{ij}$ be the flow on arc $(i,j)$ for $(i,j) \\in A$."
 ]
},
{
 "cell_type": "code",
 "execution_count": 3,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Defining variables for the first time\n"
   ]
  }
 ],
 "source": [
  "try:\n",
  "    model.del_component(model.x)\n",
  "    model.del_component(model.x_index)\n",
  "    model.del_component(model.x_index_0)\n",
  "    model.del_component(model.x_index_1)\n",
  "except:\n",
  "    print(\"Defining variables for the first time\")\n",
  "model.x = Var(A, domain=Binary, bounds=mybounds)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 4,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Defining variables for first time\n"
   ]
  }
 ],
 "source": [
  "try: \n",
  "    model.del_component(model.y)\n",
```

```
    "    model.del_component(model.y_index)\n",
    "    model.del_component(model.y_index_0)\n",
    "    model.del_component(model.y_index_1)\n",
    "except:\n",
    "    print(\"Defining variables for first time\")\n",
    "model.y = Var(A,A, domain=Binary, bounds=mybounds2)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Defining objective for the first time\n"
     ]
    }
   ],
   "source": [
    "try:\n",
    "    model.del_component(model.Moves_Distance)\n",
    "except:\n",
    "    print(\"Defining objective for the first time\")\n",
    "model.Moves_Distance = Objective(expr=sum(m[i,j]*model.x[i,j] for (i,j) in A)+sum(sum(sum(sum(d[j,l] * model.y[i,j,k,l] for i in W) for j in S) for k in W) for l in S), sense=minimize)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Flow Balance constraints:\n",
    "\n",
    "$$\n",
    "\\sum_{j \\in V: (i,j) \\in A} x_{ij} - \\sum_{j \\in V: (j,i) \\in A} x_{ji} = b_i, i \\in V \\mbox{ Flow Balance}\n",
    "$$"
   ]
```

```
 },
 {
  "cell_type": "code",
  "execution_count": 6,
  "metadata": {
   "scrolled": true
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Defining constraints for the first time\n"
    ]
   }
  ],
  "source": [
   "try:\n",
   "    model.del_component(model.FlowBalance)\n",
   "    model.del_component(model.FlowBalance_index)\n",
   "except:\n",
   "    print(\"Defining constraints for the first time\")\n",
   "    \n",
   "model.FlowBalance = ConstraintList()\n",
   "for i in V:\n",
   "    if i in W:\n",
   "        model.FlowBalance.add(sum(model.x[i,j] for j in V if (i,j) in A) - sum(model.x[j,i] for j in V if (j,i) in A)
== b[i])\n",
   "    else:\n",
   "        model.FlowBalance.add(sum(model.x[i,j] for j in V if (i,j) in A) - sum(model.x[j,i] for j in V if (j,i) in A)
>= b[i])\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 7,
  "metadata": {},
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Defining constraints for the first time\n"
    ]
   }
  ],
  "source": [
   "try:\n",
```

```
    "    model.del_component(model.Flip_on)\n",
    "    model.del_component(model.Flip_on_index)\n",
    "except:\n",
    "    print(\"Defining constraints for the first time\")\n",
    "    \n",
    "model.Flip_on = ConstraintList()\n",
    "for i in W:\n",
    "    for j in S:\n",
    "        for k in W:\n",
    "            for l in S:\n",
    "                if team_worker[i] == team_worker[k]: model.Flip_on.add(model.x[i,j] + model.x [k,l] <= 
model.y[i,j,k,l] + 1)\n",
    "                else: model.Flip_on.add(model.y[i,j,k,l] == 0)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Defining constraints for the first time\n"
     ]
    }
   ],
   "source": [
    "try:\n",
    "    model.del_component(model.RankConstraint)\n",
    "    model.del_component(model.RankConstraint_index)\n",
    "except:\n",
    "    print(\"Defining constraints for the first time\")\n",
    "    \n",
    "model.RankConstraint = ConstraintList()\n",
    "for i in W:\n",
    "    for j in S:\n",
    "        if rank_worker[i] != rank_space [j]: model.RankConstraint.add(model.x[i,j] ==0)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Specify solver and solve."
   ]
  },
```

```
{
 "cell_type": "code",
 "execution_count": 9,
 "metadata": {},
 "outputs": [],
 "source": [
  "solver = SolverFactory(\"glpk\")\n",
  "status = solver.solve(model)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 10,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Status = \n",
    "Problem: \n",
    "- Name: unknown\n",
    "  Lower bound: 1240.0\n",
    "  Upper bound: 1240.0\n",
    "  Number of objectives: 1\n",
    "  Number of constraints: 78663\n",
    "  Number of variables: 78681\n",
    "  Number of nonzeros: 138493\n",
    "  Sense: minimize\n",
    "Solver: \n",
    "- Status: ok\n",
    "  Termination condition: optimal\n",
    "  Statistics: \n",
    "    Branch and bound: \n",
    "      Number of bounded subproblems: 131\n",
    "      Number of created subproblems: 131\n",
    "  Error rc: 0\n",
    "  Time: 0.8706469535827637\n",
    "Solution: \n",
    "- number of solutions: 0\n",
    "  number of solutions displayed: 0\n",
    "\n"
   ]
  }
 ],
 "source": [
  "print(\"Status = %s\" % status)"
 ]
```

```
},
{
 "cell_type": "code",
 "execution_count": 11,
 "metadata": {
  "scrolled": true
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "x[1,ACO] = 0.000000\n",
    "x[1,A2W1] = 0.000000\n",
    "x[1,A3W] = 0.000000\n",
    "x[1,AEC] = 0.000000\n",
    "x[1,AIC] = 0.000000\n",
    "x[1,A2W2] = 0.000000\n",
    "x[1,BCO] = 0.000000\n",
    "x[1,B2W1] = 0.000000\n",
    "x[1,B3W] = 1.000000\n",
    "x[1,BEC] = 0.000000\n",
    "x[1,BIC] = 0.000000\n",
    "x[1,BIO1] = 0.000000\n",
    "x[1,BIO2] = 0.000000\n",
    "x[1,B2W2] = 0.000000\n",
    "x[2,ACO] = 0.000000\n",
    "x[2,A2W1] = 0.000000\n",
    "x[2,A3W] = 0.000000\n",
    "x[2,AEC] = 0.000000\n",
    "x[2,AIC] = 0.000000\n",
    "x[2,A2W2] = 0.000000\n",
    "x[2,BCO] = 0.000000\n",
    "x[2,B2W1] = 0.000000\n",
    "x[2,B3W] = 0.000000\n",
    "x[2,BEC] = 1.000000\n",
    "x[2,BIC] = 0.000000\n",
    "x[2,BIO1] = 0.000000\n",
    "x[2,BIO2] = 0.000000\n",
    "x[2,B2W2] = 0.000000\n",
    "x[3,ACO] = 0.000000\n",
    "x[3,A2W1] = 0.000000\n",
    "x[3,A3W] = 0.000000\n",
    "x[3,AEC] = 0.000000\n",
    "x[3,AIC] = 0.000000\n",
    "x[3,A2W2] = 0.000000\n",
    "x[3,BCO] = 0.000000\n",
    "x[3,B2W1] = 0.000000\n",
```

"x[3,B3W] = 0.000000\n",
"x[3,BEC] = 1.000000\n",
"x[3,BIC] = 0.000000\n",
"x[3,BIO1] = 0.000000\n",
"x[3,BIO2] = 0.000000\n",
"x[3,B2W2] = 0.000000\n",
"x[4,ACO] = 0.000000\n",
"x[4,A2W1] = 0.000000\n",
"x[4,A3W] = 0.000000\n",
"x[4,AEC] = 0.000000\n",
"x[4,AIC] = 0.000000\n",
"x[4,A2W2] = 0.000000\n",
"x[4,BCO] = 0.000000\n",
"x[4,B2W1] = 0.000000\n",
"x[4,B3W] = 0.000000\n",
"x[4,BEC] = 0.000000\n",
"x[4,BIC] = 1.000000\n",
"x[4,BIO1] = 0.000000\n",
"x[4,BIO2] = 0.000000\n",
"x[4,B2W2] = 0.000000\n",
"x[5,ACO] = 0.000000\n",
"x[5,A2W1] = 0.000000\n",
"x[5,A3W] = 0.000000\n",
"x[5,AEC] = 0.000000\n",
"x[5,AIC] = 0.000000\n",
"x[5,A2W2] = 0.000000\n",
"x[5,BCO] = 0.000000\n",
"x[5,B2W1] = 0.000000\n",
"x[5,B3W] = 0.000000\n",
"x[5,BEC] = 0.000000\n",
"x[5,BIC] = 0.000000\n",
"x[5,BIO1] = 1.000000\n",
"x[5,BIO2] = 0.000000\n",
"x[5,B2W2] = 0.000000\n",
"x[6,ACO] = 0.000000\n",
"x[6,A2W1] = 0.000000\n",
"x[6,A3W] = 0.000000\n",
"x[6,AEC] = 0.000000\n",
"x[6,AIC] = 0.000000\n",
"x[6,A2W2] = 0.000000\n",
"x[6,BCO] = 0.000000\n",
"x[6,B2W1] = 0.000000\n",
"x[6,B3W] = 0.000000\n",
"x[6,BEC] = 1.000000\n",
"x[6,BIC] = 0.000000\n",
"x[6,BIO1] = 0.000000\n",
"x[6,BIO2] = 0.000000\n",
"x[6,B2W2] = 0.000000\n",

"x[7,ACO] = 0.000000\n",
"x[7,A2W1] = 0.000000\n",
"x[7,A3W] = 0.000000\n",
"x[7,AEC] = 0.000000\n",
"x[7,AIC] = 0.000000\n",
"x[7,A2W2] = 0.000000\n",
"x[7,BCO] = 0.000000\n",
"x[7,B2W1] = 0.000000\n",
"x[7,B3W] = 0.000000\n",
"x[7,BEC] = 1.000000\n",
"x[7,BIC] = 0.000000\n",
"x[7,BIO1] = 0.000000\n",
"x[7,BIO2] = 0.000000\n",
"x[7,B2W2] = 0.000000\n",
"x[8,ACO] = 0.000000\n",
"x[8,A2W1] = 0.000000\n",
"x[8,A3W] = 0.000000\n",
"x[8,AEC] = 0.000000\n",
"x[8,AIC] = 0.000000\n",
"x[8,A2W2] = 0.000000\n",
"x[8,BCO] = 0.000000\n",
"x[8,B2W1] = 0.000000\n",
"x[8,B3W] = 0.000000\n",
"x[8,BEC] = 0.000000\n",
"x[8,BIC] = 1.000000\n",
"x[8,BIO1] = 0.000000\n",
"x[8,BIO2] = 0.000000\n",
"x[8,B2W2] = 0.000000\n",
"x[9,ACO] = 0.000000\n",
"x[9,A2W1] = 0.000000\n",
"x[9,A3W] = 0.000000\n",
"x[9,AEC] = 0.000000\n",
"x[9,AIC] = 0.000000\n",
"x[9,A2W2] = 0.000000\n",
"x[9,BCO] = 0.000000\n",
"x[9,B2W1] = 0.000000\n",
"x[9,B3W] = 0.000000\n",
"x[9,BEC] = 1.000000\n",
"x[9,BIC] = 0.000000\n",
"x[9,BIO1] = 0.000000\n",
"x[9,BIO2] = 0.000000\n",
"x[9,B2W2] = 0.000000\n",
"x[10,ACO] = 0.000000\n",
"x[10,A2W1] = 0.000000\n",
"x[10,A3W] = 0.000000\n",
"x[10,AEC] = 0.000000\n",
"x[10,AIC] = 0.000000\n",
"x[10,A2W2] = 0.000000\n",

"x[10,BCO] = 0.000000\n",
"x[10,B2W1] = 0.000000\n",
"x[10,B3W] = 0.000000\n",
"x[10,BEC] = 0.000000\n",
"x[10,BIC] = 1.000000\n",
"x[10,BIO1] = 0.000000\n",
"x[10,BIO2] = 0.000000\n",
"x[10,B2W2] = 0.000000\n",
"x[11,ACO] = 0.000000\n",
"x[11,A2W1] = 0.000000\n",
"x[11,A3W] = 1.000000\n",
"x[11,AEC] = 0.000000\n",
"x[11,AIC] = 0.000000\n",
"x[11,A2W2] = 0.000000\n",
"x[11,BCO] = 0.000000\n",
"x[11,B2W1] = 0.000000\n",
"x[11,B3W] = 0.000000\n",
"x[11,BEC] = 0.000000\n",
"x[11,BIC] = 0.000000\n",
"x[11,BIO1] = 0.000000\n",
"x[11,BIO2] = 0.000000\n",
"x[11,B2W2] = 0.000000\n",
"x[12,ACO] = 0.000000\n",
"x[12,A2W1] = 0.000000\n",
"x[12,A3W] = 0.000000\n",
"x[12,AEC] = 1.000000\n",
"x[12,AIC] = 0.000000\n",
"x[12,A2W2] = 0.000000\n",
"x[12,BCO] = 0.000000\n",
"x[12,B2W1] = 0.000000\n",
"x[12,B3W] = 0.000000\n",
"x[12,BEC] = 0.000000\n",
"x[12,BIC] = 0.000000\n",
"x[12,BIO1] = 0.000000\n",
"x[12,BIO2] = 0.000000\n",
"x[12,B2W2] = 0.000000\n",
"x[13,ACO] = 0.000000\n",
"x[13,A2W1] = 0.000000\n",
"x[13,A3W] = 0.000000\n",
"x[13,AEC] = 0.000000\n",
"x[13,AIC] = 1.000000\n",
"x[13,A2W2] = 0.000000\n",
"x[13,BCO] = 0.000000\n",
"x[13,B2W1] = 0.000000\n",
"x[13,B3W] = 0.000000\n",
"x[13,BEC] = 0.000000\n",
"x[13,BIC] = 0.000000\n",
"x[13,BIO1] = 0.000000\n",

"x[13,BIO2] = 0.000000\n",
"x[13,B2W2] = 0.000000\n",
"x[14,ACO] = 0.000000\n",
"x[14,A2W1] = 0.000000\n",
"x[14,A3W] = 0.000000\n",
"x[14,AEC] = 1.000000\n",
"x[14,AIC] = 0.000000\n",
"x[14,A2W2] = 0.000000\n",
"x[14,BCO] = 0.000000\n",
"x[14,B2W1] = 0.000000\n",
"x[14,B3W] = 0.000000\n",
"x[14,BEC] = 0.000000\n",
"x[14,BIC] = 0.000000\n",
"x[14,BIO1] = 0.000000\n",
"x[14,BIO2] = 0.000000\n",
"x[14,B2W2] = 0.000000\n",
"x[15,ACO] = 0.000000\n",
"x[15,A2W1] = 0.000000\n",
"x[15,A3W] = 0.000000\n",
"x[15,AEC] = 0.000000\n",
"x[15,AIC] = 1.000000\n",
"x[15,A2W2] = 0.000000\n",
"x[15,BCO] = 0.000000\n",
"x[15,B2W1] = 0.000000\n",
"x[15,B3W] = 0.000000\n",
"x[15,BEC] = 0.000000\n",
"x[15,BIC] = 0.000000\n",
"x[15,BIO1] = 0.000000\n",
"x[15,BIO2] = 0.000000\n",
"x[15,B2W2] = 0.000000\n",
"x[16,ACO] = 0.000000\n",
"x[16,A2W1] = 0.000000\n",
"x[16,A3W] = 0.000000\n",
"x[16,AEC] = 1.000000\n",
"x[16,AIC] = 0.000000\n",
"x[16,A2W2] = 0.000000\n",
"x[16,BCO] = 0.000000\n",
"x[16,B2W1] = 0.000000\n",
"x[16,B3W] = 0.000000\n",
"x[16,BEC] = 0.000000\n",
"x[16,BIC] = 0.000000\n",
"x[16,BIO1] = 0.000000\n",
"x[16,BIO2] = 0.000000\n",
"x[16,B2W2] = 0.000000\n",
"x[17,ACO] = 0.000000\n",
"x[17,A2W1] = 0.000000\n",
"x[17,A3W] = 0.000000\n",
"x[17,AEC] = 1.000000\n",

"x[17,AIC] = 0.000000\n",
"x[17,A2W2] = 0.000000\n",
"x[17,BCO] = 0.000000\n",
"x[17,B2W1] = 0.000000\n",
"x[17,B3W] = 0.000000\n",
"x[17,BEC] = 0.000000\n",
"x[17,BIC] = 0.000000\n",
"x[17,BIO1] = 0.000000\n",
"x[17,BIO2] = 0.000000\n",
"x[17,B2W2] = 0.000000\n",
"x[18,ACO] = 0.000000\n",
"x[18,A2W1] = 0.000000\n",
"x[18,A3W] = 0.000000\n",
"x[18,AEC] = 1.000000\n",
"x[18,AIC] = 0.000000\n",
"x[18,A2W2] = 0.000000\n",
"x[18,BCO] = 0.000000\n",
"x[18,B2W1] = 0.000000\n",
"x[18,B3W] = 0.000000\n",
"x[18,BEC] = 0.000000\n",
"x[18,BIC] = 0.000000\n",
"x[18,BIO1] = 0.000000\n",
"x[18,BIO2] = 0.000000\n",
"x[18,B2W2] = 0.000000\n",
"x[19,ACO] = 0.000000\n",
"x[19,A2W1] = 0.000000\n",
"x[19,A3W] = 0.000000\n",
"x[19,AEC] = 0.000000\n",
"x[19,AIC] = 1.000000\n",
"x[19,A2W2] = 0.000000\n",
"x[19,BCO] = 0.000000\n",
"x[19,B2W1] = 0.000000\n",
"x[19,B3W] = 0.000000\n",
"x[19,BEC] = 0.000000\n",
"x[19,BIC] = 0.000000\n",
"x[19,BIO1] = 0.000000\n",
"x[19,BIO2] = 0.000000\n",
"x[19,B2W2] = 0.000000\n",
"x[20,ACO] = 0.000000\n",
"x[20,A2W1] = 0.000000\n",
"x[20,A3W] = 0.000000\n",
"x[20,AEC] = 1.000000\n",
"x[20,AIC] = 0.000000\n",
"x[20,A2W2] = 0.000000\n",
"x[20,BCO] = 0.000000\n",
"x[20,B2W1] = 0.000000\n",
"x[20,B3W] = 0.000000\n",
"x[20,BEC] = 0.000000\n",

```
    "x[20,BIC] = 0.000000\n",
    "x[20,BIO1] = 0.000000\n",
    "x[20,BIO2] = 0.000000\n",
    "x[20,B2W2] = 0.000000\n",
    "Moves_Distance = 1240.000000\n",
    "total_moves = 10\n",
    "total_distance = 1230 ft.\n"
   ]
  },
  {
   "ename": "TypeError",
   "evalue": "int() argument must be a string, a bytes-like object or a number, not 'generator'",
   "output_type": "error",
   "traceback": [
    "\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",
    "\u001b[1;31mTypeError\u001b[0m                                 Traceback (most recent call last)",
    "\u001b[1;32m<ipython-input-11-ade1d3fcf410>\u001b[0m in
\u001b[0;36m<module>\u001b[1;34m\u001b[0m\n\u001b[0;32m      7\u001b[0m
\u001b[0mprint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m\"total_moves = %i\"\u001b[0m
\u001b[1;33m%\u001b[0m
\u001b[0mtotal_moves\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b
[0m\u001b[0m\n\u001b[0;32m      8\u001b[0m
\u001b[0mprint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m\"total_distance = %i ft.\"\u001b[0m
\u001b[1;33m%\u001b[0m
\u001b[0mtotal_distance\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001
b[0m\u001b[0m\n\u001b[1;32m----> 9\u001b[1;33m \u001b[0mteam_distance\u001b[0m
\u001b[1;33m=\u001b[0m
\u001b[0mint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0msum\u001b[0m\u001b[0m\u0
01b[1;33m(\u001b[0m\u001b[0md\u001b[0m\u001b[1;33m[\u001b[0m\u001b[0mj\u001b[0m\u001b[1;33
m,\u001b[0m\u001b[0ml\u001b[0m\u001b[1;33m]\u001b[0m \u001b[1;33m*\u001b[0m
\u001b[0mvalue\u001b[0m\u001b[1;33m(\u001b[0m\u001b[0mmodel\u001b[0m\u001b[1;33m.\u001b[0
m\u001b[0my\u001b[0m\u001b[1;33m[\u001b[0m\u001b[0mi\u001b[0m\u001b[1;33m,\u001b[0m\u001b
[0mj\u001b[0m\u001b[1;33m,\u001b[0m\u001b[0mk\u001b[0m\u001b[1;33m,\u001b[0m\u001b[0ml\u00
1b[0m\u001b[1;33m]\u001b[0m \u001b[1;32mfor\u001b[0m \u001b[0mi\u001b[0m
\u001b[1;32min\u001b[0m \u001b[0mW\u001b[0m\u001b[1;33m)\u001b[0m \u001b[1;32mfor\u001b[0m
\u001b[0mj\u001b[0m \u001b[1;32min\u001b[0m \u001b[0mS\u001b[0m\u001b[1;33m)\u001b[0m
\u001b[1;32mfor\u001b[0m \u001b[0mk\u001b[0m \u001b[1;32min\u001b[0m
\u001b[0mW\u001b[0m\u001b[1;33m)\u001b[0m \u001b[1;32mfor\u001b[0m \u001b[0ml\u001b[0m
\u001b[1;32min\u001b[0m
\u001b[0mS\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u001b[0m\u001b
[0m\n\u001b[0m\u001b[0;32m     10\u001b[0m
\u001b[0mprint\u001b[0m\u001b[1;33m(\u001b[0m\u001b[1;34m\"Team %i total distance: %i
ft.\"\u001b[0m \u001b[1;33m%\u001b[0m
\u001b[0mteam_distance\u001b[0m\u001b[1;33m)\u001b[0m\u001b[1;33m\u001b[0m\u001b[1;33m\u00
1b[0m\u001b[0m\n\u001b[0;32m     11\u001b[0m \u001b[1;33m\u001b[0m\u001b[0m\n",
    "\u001b[1;31mTypeError\u001b[0m: int() argument must be a string, a bytes-like object or a number,
not 'generator'"
   ]
```

```
   }
  ],
  "source": [
   "for (i,j) in A:\n",
   "    print(\"%s = %f\" % (model.x[i,j], value(model.x[i,j])))\n",
   "\n",
   "print(\"Moves_Distance = %f\" % value(model.Moves_Distance))\n",
   "total_moves = int(sum(m[i,j]* value(model.x[i,j] )for (i,j) in A))\n",
   "total_distance = int(value(model.Moves_Distance) - total_moves)\n",
   "print(\"total_moves = %i\" % total_moves)\n",
   "print(\"total_distance = %i ft.\" % total_distance)\n",
   "team_distance = int((sum(d[j,l] * value(model.y[i,j,k,l] for i in W) for j in S) for k in W) for l in S)\n",
   "print(\"Team %i total distance: %i ft.\" % team_distance)\n",
   "\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": []
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Python 3",
  "language": "python",
  "name": "python3"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.7.7"
 }
},
"nbformat": 4,
"nbformat_minor": 4
}
```