**IE 517 HW 1**

**Alican Yılmaz(2016402093)**

**May 2021**

**1.Problem Definition:**

The aim of the work is developing construction and improvement heuristics for a “generalized assignment problem”. The given problem is cost minimization version of generalized assignment problem and the test instances with their optimal objective values are given. For the following sections, ci,j refers to cost of allocating job j to agent i, and ri,j refers to resource consumed in allocating job j to agent i. In section 2, construction heuristic is explained, and the pseudocode is given. In section 3, Improvement heuristic is explained and the pseudocode of it is given. Finally, the outputs of the heuristics developed for each instance are given in section 4.

**2. Construction Heuristic**

The problem is suitable for various construction heuristic ideas; however, one must also consider the feasibility issue for each of the heuristics developed. In this part, the heuristic which gives a feasible solution for the test instances, i.e., all the jobs must be assigned to an agent without violating the capacity constraints, is selected and implemented for the problem. The construction heuristic developed is also taken as first step of the improvement heuristic developed in the Section 3, as a way of obtaining an initial feasible solution.

The idea of the heuristic is as follows:

1. Normalize the costs and resources of assigning job i to job j for each i,j.
2. Define fi,j=normalized(ci,j)+alpha\*normalized(ri,j).
3. Find fi,j values of each i,j pair and sort them in increasing order of fi,j.
4. Do the assignment from the start considering the capacity constraints of the agents.

Here alpha is a pre-defined parameter and must be taken as small as possible. (For example alpha=0 would mean “do the assignment ONLY taking into account of the costs. However, this will most likely give infeasible solution as resources are not considered at all. Similarly choosing alpha infinite would give inferior result in the expense of finding a feasible solution, as it takes into account the resources, but not the costs.) For each instances several alpha values are tried starting from 0.1 and increasing incrementally by 0.1. At the end, the optimal alpha is obtained and used for the construction heuristic. The process of selecting alpha can be automatized but it is omitted in this work, rather the best-found alpha is given for each of the instances. Note that, alpha is only used in the sorting step and as a constant.

Normalization is done to prevent the unit differences between the resources and costs. For example, normalized(ci,j) =ci,j/max(ci,j) and normalized(ri,j)=ri,j/max(ri,j).

**Text

Description automatically generated**

*Figure-1 Solution Representation*

Above, the solution representation can be seen. Keys of the dictionary refers to the agents, and values of the dictionary refers to the jobs assigned to the corresponding agent. Dictionary representation is preferred over list to keep track of the agents easily.

**2.1. Pseudo Code of the Algorithm**

**Step 1. Initialization**

* 1. Input the value of parameter alpha
  2. Calculate the normalized costs and normalized resources of each (i,j)
  3. Find fi,j values of each i,j pair and sort the pairs in increasing order of fi,j and keep them in sorted dictionary.
  4. Initialize remaining capacity, total cost and assigned jobs.
  5. S<- ∅ // S refers to solution

**Step 2.** **For** all (i,j) in sorted dictionary **do** the following:

2.1. **If** remaining capacity>resource(i,j) and j **not in** assigned jobs:

* + 1. Update the total cost, assigned jobs and remaining capacity
    2. Call ﻿delete\_job\_from\_dict() // This function is used to check whether there is any job which is not assigned at the end of the heuristic.
    3. Update S

2.2. **Else:** continue

1. **Improvement Heuristic**

In this part, an improvement heuristic is developed for the defined problem. This heuristic consists of two phases: “Finding initial feasible solution” and “Improvement of the initial solution”. As for the first phase, the construction heuristic developed in Section 2 is implemented to obtain a feasible initial solution. Different alpha values are given to obtain different initial feasible solutions because “multi-starting” is assumed to improve the objective in that heuristic. An example of a solution representation is shown in *Figure-1.* Local search idea (similar to steepest descent) is used in this problem with swap operator. Swap operator chooses two jobs from two agents and swap their assignments. For example, in the example of *Figure-1*, one swap could be exchanging the agent of job 11 with job 6. The local search is stopped when no new better solution is found in the neighborhood space of the current solution. At each iteration, current solution is updated, and incumbent solution/best cost is kept tracked. Several functions such as swap\_jobs(), check\_capacity() are defined for ease of implementation. Detailed explanation of those can be found on the comment sections of the code.

* 1. **Pseudo Code of the Algorithm**

**Phase 1: The Construction Heuristic**

**Step 1. Initialization**

* 1. Input the value of parameter alpha
  2. Calculate the normalized costs and normalized resources of each (i,j)
  3. Find fi,j values of each i,j pair and sort the pairs in increasing order of fi,j and keep them in sorted dictionary.
  4. Initialize remaining capacity, total cost and assigned jobs.
  5. S<- ∅ // S refers to solution

**Step 2.** **For** all (i,j) in sorted dictionary **do** the following:

2.1. **If** remaining capacity>resource(i,j) and j **not in** assigned jobs:

2.1.1. Update the total cost, assigned jobs and remaining capacity

2.1.2. Call ﻿delete\_job\_from\_dict() // This function is used to check whether there is any job not assigned at the end of the heuristic.

2.1.3. Update S

2.2. **Else:** continue

**Phase 2: Local Search**

**Step 1. Initialization**

* 1. best cost<- total cost // best cost is assigned to cost of the initial solution obtained from constructed heuristic
  2. current solution <-S //current solution is assigned to initial solution obtained from constructed heuristic
  3. incumbent solution <- current solution

**Step 2. While** the solution is improved **do** the following**:**

2.1. **for** each job (i,j) pair **do** the following:

2.1.1. neighbor solution <-swap\_jobs(i,j, current solution) // calls the swap function, which swaps the agents of the job i,j and returns the swapped solution

2.1.2. cost<-calculate\_cost(neighbor solution) // calls the calculate\_cost function which returns the cost of the given solution

2.1.3. **if** cost<best cost **and** check\_capacity(neighbour solution) is *True*: // checks if the new solution is better and feasible

2.1.3.1. best cost<-cost

2.1.3.2. incumbent solution<- neighbor solution

2.2. current solution<- incumbent solution

**4.Outputs**

*Table 1 Results of construction heuristic*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | Solution (jobs assigned to each agent) | Obj.  Value  Construction Heuristic | % Gap from optimal objective value | CPU Time (s) |
| 5-25 | ﻿{1: [11, 20, 4, 12, 16],  2: [6, 13, 3, 17, 2, 5, 14],  3: [1, 24, 19, 22, 15],  4: [21, 8, 10, 7],  5: [9, 25, 23, 18]} | 452 | 3.20 | ﻿0.001 |
| 8-40 | ﻿{1: [37, 16, 14, 28, 4, 3],  2: [30, 9, 2],  3: [26, 6, 34, 20],  4: [27, 18, 22, 8, 5],  5: [39, 7, 33],  6: [23, 13, 25, 29, 24, 31],  7: [10, 32, 11, 40, 35, 38, 21],  8: [36, 12, 17, 15, 1, 19]} | 662 | 2.48 | 0.005 |
| 10-50 | ﻿{1: [24, 25, 29, 39],  2: [27, 1, 20, 28],  3: [21, 9, 23, 50, 13, 17, 36],  4: [3, 2, 46, 19, 38, 14],  5: [15, 31, 12, 11, 41, 26, 4],  6: [7, 32, 43, 49, 5],  7: [45, 34, 30, 47],  8: [22, 6, 37],  9: [35, 10, 40, 8, 48],  10: [33, 42, 18, 44, 16]} | 587 | 2.44 | 0.011 |

*Table 2 Results of improvement heuristic*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance | Solution (jobs assigned to each agent) | Obj.  Value  Improvement Heuristic | % Gap from optimal objective value | CPU Time (s) |
| 5-25 | ﻿{ ﻿{1: [11, 20, 4, 16, 22],  2: [6, 13, 3, 17, 2, 5, 14],  3: [1, 24, 19, 15, 12],  4: [21, 8, 10, 7],  5: [9, 25, 23, 18]} | 446 | 1.83 | ﻿0.026 |
| 8-40 | ﻿{1: [37, 16, 14, 5, 4],  2: [27, 30, 31, 9],  3: [26, 13, 24, 20, 28],  4: [22, 18, 8, 1],  5: [39, 34, 19, 3, 2],  6: [23, 25, 29, 33, 7],  7: [10, 32, 11, 40, 35, 38, 21],  8: [36, 6, 17, 15, 12]} | 662 | 2.48 | 0.06 |
| 10-50 | ﻿{1: [24, 25, 29, 39],  2: [27, 1, 20, 28],  3: [21, 9, 23, 50, 13, 17, 36],  4: [3, 2, 46, 19, 38, 14],  5: [15, 31, 12, 11, 41, 26, 4],  6: [7, 32, 43, 49, 5],  7: [45, 34, 30, 47],  8: [22, 6, 37],  9: [35, 10, 40, 8, 48],  10: [33, 42, 18, 44, 16]} | 587 | 2.44 | 0.11 |

1. **Conclusion and Further Remarks**

As can be seen from the table, developed construction heuristic gives reasonably good results in a very short time. (biggest having %3.2 gap from the objective.). Thus, improvement heuristic with local search idea and given neighborhood structure only improved the first solution from 452 to 446. The other two got stuck in the local optimum, with the current setting at least. Different initial solutions are tried by changing the alpha value in the construction phase, to escape from local optimum. Still the objective did not get better. The best results obtained are shown in the Table-1 and Table-2.

To improve the proposed heuristics, another “move operator” in addition to swap operator, which takes a job from the assigned agent and assign it to a different agent, could be added. This way, the neighborhood space could get bigger and better solutions may be found. Simulated annealing idea could also be implemented as improvement heuristic, which enables inferior results to be selected with some probability in the solution search. Another idea could be, instead of selecting the best neighbor as current solution ,i.e. the one which gives smallest total cost, it could be selected from a pool of neighbors with a weighted probability of their corresponding costs. This would allow the inferior result to be selected as current solution and may help to escape from local optimum (diversification). However, for this problem setting the proposed heuristics developed in Section-2 and Section-3 are found to be satisfactory enough.