Complex Task Allocation: A Comparison of Metaheuristics

Group 27

Lucas Wojciechowksi, Ariel Weingarten, Alexander Maguire, Austin Dobrik, Dane Carr

July 21, 2014

Summary

1 Introduction

2 Literature Review

3 Problem Formulation and Modeling

Suppose we have a number of mobile *robot*s

$$robots = \{robot_i | i \in [1, n_{robots}]\}$$

Using these *robot*s, we are to perform a number of tasks

$$tasks = \{task_i | i \in [1, n_{tasks}]\}$$

Each task has a *priority*

$$\text{priority}(task_i) \in [0, 1]$$

*robot*s are not homogeneous; some are more "skilled" at certain tasks than others

$$\text{skill}(robot_i, task_j) \in [0, 1]$$

Tasks are located some distance each other

$$distance(task_i, task_j) > 0 \in \mathbb{R}$$

Each *robot* starts at some location, its *depot*. $robot_i$ starts at $depot_i$

Each *depot* is located a certain distance from each task.

$$distance(depot_i, task_j)$$

*task*s are assigned to *robot*s.

$$task_j \rightarrow robot_i$$

We seek, for each *robot*, a path through the *task*s such that each *task* is performed exactly once and our overall cost is minimized.

A solution, S, consists of a mapping of *robot*s to paths through *task*s.

$$S(robot_i) = \{task_0, task_1, ..., task_n\}$$

3.0.1 Cost Function

The cost of some path, $S(i)$ is given by

$$\text{cost}(S(i)) = w(i, S(i, |S(i)|), S(i, 0)) + \sum_{t=0}^{|S(i)|-1} w(i, S(i,t), S(i, t+1))$$

Subject to the constraint

$$n \notin S(i) \quad \text{if} \quad M_{skill}(i, n) = 0$$

Therefore, the overall performance of a solution is:

$$\sum_{i \in robot} \text{cost}_i$$

Our weighted-distance function is defined as follows:

$$w(i, \mathit{src}, \mathit{dst}) = \begin{cases} (\alpha - \text{priority}(\mathit{dst}))^{\gamma} \times (\beta - \text{skill}(i, \mathit{dst})) \times \text{distance}(\mathit{src}, \mathit{dst}) & \mathit{dst} \neq S(i, 0) \\ \text{distance}(\mathit{src}, \mathit{dst}) & \text{otherwise} \end{cases}$$

where $\alpha, \beta \geq 1$. These parameters are to ensure that the priority and skill factors are aligned with the minimization of the function.

γ is used to tweak the relative importance of a task's priority, and has no constraints on it.

The distance(src, dst) function makes use of $D_{\text{task-task}}$, $D_{\text{depot-task}}$, and $D_{\text{task-depot}}$. $D_{\text{task-task}}$ is a square matrix that contains the distances between tasks i.e. the distance between task i and j is found at $D_{\text{task-task}}(i, j)$. $D_{\text{depot-task}}$ contains the distances from depots to tasks i.e. the distance from depot i to task j is $D_{\text{task-task}}(i, j)$. $D_{\text{task-depot}}$ contains the distances from the tasks to the depots i.e. the distance from task i to depot j is $D_{\text{task-depot}}(i, j)$

The priority(dst) function makes use of P. P is a vector where the priority of task$_i$ is contained in $P(i)$.

The skill(i, dst) function makes use of the matrix M_{skill}. The skill of robot$_i$ at performing task$_j$ is contained in $M_{skill}(i, j)$

3.1 Modeling as Traveling Salesman Problem

Given a population of robots R, a set of tasks T, a priority for each task P, each robot's skill at performing each task $M_{skill}(r, t)$ and an ordered assignment of tasks to robots S_r, for each $r \in R$ we can create a directed, weighted graph as follows:

We will model complex task allocation loosely on the traveling salesman problem.

Given a random initial solution, S_0,

$$\begin{aligned} G &= (V, E) \\ V &= \{start\} \cup T \\ E_{x,t} &= -P_a S_{r,t} \qquad \forall t \in A_r \\ E_{x,start} &= 0 \end{aligned}$$

Which is to say, for each robot we create a graph with a start node and a node for each task assigned to said robot. The cost of traveling to any node is ¡¡INSERT COST FUNCTION DESCRIPTION HERE¿¿. The cost of traveling to the start node is the distance from the current node to the start node.

From here, we can run TSP on the created graph, with the robot starting at the start node. It is trivial to show that regardless of the path taken, the total cost traveled is:

$$C_r = \sum_t^{A_r} -P_t S_{r,t}$$

In order to maximize efficiency of the system as a whole, we must then minimize the total cost for all robots:

$$C = \sum_r^R C_r = \sum_r^R \sum_t^{A_r} -P_t S_{r,t}$$

4 Potential other model

This is almost verbatim out of the literature.

$$x_{ij} = \begin{cases} 1 & \text{if arc(i,j) is used in the tour} \\ 0 & otherwise \end{cases} \tag{1}$$

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \times x_{ij} \tag{2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} = n \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |\text{SubTour}| - 1 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A \tag{5}$$

$$\forall \text{SubTour} \subseteq V \backslash \phi \tag{6}$$

4.1 Reduced Size Problem

In this section, we define a reduced size complex task allocation problem which will be used to perform hand-iterations of various optimization algorithms in the following sections.

Suppose we have 2 robots

$$robots = \{robot_0, robot_1\}$$

These robots must collectively perform 3 tasks

$$tasks = \{task_0, task_1, task_2\}$$

The priority of each task is

$task_0$	0.1
$task_1$	0.2
$task_2$	0.3

The skill of each robot at each task is

	$task_0$	$task_1$	$task_2$
$robot_0$	0.1	0.4	0.5
$robot_1$	0.2	0.3	0.6

The distances between tasks are

	$task_0$	$task_1$	$task_2$
$task_0$	0	0.1	0.2
$task_1$	0.1	0	0.3
$task_2$	0.2	0.3	0

Each robot, $robot_i$, starts at a depot, $depot_i$. The distance from each depot to each task is

	$task_0$	$task_1$	$task_2$
$depot_0$	0.6	0.3	0.1
$depot_1$	0.5	0.4	0.2

4.2 Real Problem

5 Proposed Solution

As an initial solution, we will use

$$S_0 = [0, 1, sentinel, 2]$$

Which specifies the paths

$$robot_0 : depot_0 \rightarrow task_0 \rightarrow task_1 \rightarrow depot_0 robot_1 : \quad depot_1 \rightarrow task_2 \rightarrow depot_1$$

5.1 Tabu Search Algorithm

5.1.1 Hand Iterations

We will use a tabu tenure of 1.

We define our tabu list as...

5.1.1.1 Iteration 1

We begin with our randomly selected initial solution

$$S_0 = [0, 1, X, 2]$$

Next, we find all neighbouring solutions and evaluate the cost of each. No solutions are tabu yet because our tabu list is empty

	swap	**solution**	**tabu?**	**cost**
$S_{1,1}$	$1, 2$	$[1, 0, X, 2]$	no	1.0810
$S_{1,2}$	$1, 3$	$[X, 1, 0, 2]$	no	0.55200
$S_{1,3}$	$1, 4$	$[2, 1, X, 0]$	no	1.3390
$S_{1,4}$	$2, 3$	$[0, X, 1, 2]$	no	1.5940
$S_{1,5}$	$2, 4$	$[0, 2, X, 1]$	no	1.2800
$S_{1,6}$	$3, 4$	$[0, 1, 2, X]$	no	0.73900

Observe that $S_{1,6}$ has the lowest cost and is therefore the best solution, $S_{1,\text{best}}$.

Since this solution was created by swapping elements 3 and 4 in S_0, we mark those as tabu with a starting tenure of 1.

position	1	2	3	4
tenure	0	0	1	1

5.1.1.2 Iteration 2

We begin with

$$S_{1,\text{best}} = [0, 1, 2, X]$$

Next, we find all neighbouring solutions

	swap	**solution**	**tabu?**	**cost**
$S_{2,1}$	$1, 2$	$[1, 0, 2, X]$	no	0.39500
$S_{2,2}$	$1, 3$	$[2, 1, 0, X]$	yes	0.86000
$S_{2,3}$	$1, 4$	$[X, 1, 2, 0]$	yes	0.95200
$S_{2,4}$	$2, 3$	$[0, 2, 1, X]$	yes	1
$S_{2,5}$	$2, 4$	$[0, X, 2, 1]$	yes	1.7100
$S_{2,6}$	$3, 4$	$[0, 1, X, 2]$	yes	1.0900

5.1.1.3 Iteration 3

5.2 Simulated Annealing Algorithm

5.3 Genetic Algorithm

5.4 Particle Swarm Optimization Algorithm

5.5 Ant Colony Optimization Algorithm

6 Performance Evaluation

7 Conclusions & Recommendations

References