**Intelligent Systems**

**Exercise 9. Metaheuristics**

# Exercise description

The objective of this exercise is to apply the concepts and methods for Solving Problems with Metaheuristics. You must use a pseudo random number generator to produce each required number during the execution of the methods.

**Team members**

Write the student id, name, and campus of each member in a different line.

1: Carlos Hinojosa A01137566, Campus Monterrey

2: Eider Diaz A00828174, Campus Monterrey

3: Miguel Cortes A01270966, Campus Monterrey

**Problems to Solve**

**Problem 1**

The **Multiple Knapsack Problem (MKP)** is a real world subset problem and it has multiple applications in theory, as well as in practice. It also arises as a subproblem within several algorithms for more complex problems and these algorithms will further benefit from any improvement in the field of MKP. The following major applications can be mentioned as possible formulations of MKP: problems in cargo loading, cutting stock, bin-packing, budget control and financial management.

MKP can be thought as a resource allocation problem, where there are m resources (knapsacks) and n objects and every object j has a profit pj. Each resource has its own budget ci (knapsack capacity) and consumption wj of resource i by object j. The aim is maximizing the sum of the profits, while working within a limited budget.

Consider the instance of MKP defined by

n = 6;

m = 2;

(Pj) = (110, 150, 70, 80, 30, 5);

(Wj) = (40, 60, 30, 40, 20, 5);

(Ci) = (65, 85).

Examples:

(S1: {5, 3}, S2: {1, 6}), weights = (S1 : 50, S2: 45), profit = 215

(S1: {2}, S2: {3, 4}), weights = (S1: 60, S2: 70), profit = 300

**Problem 2**

This is a **Simple Arithmetic Problem (SAP)** where the algorithm finds three numbers that add up to a target value.

Consider the instance of the SAP defined by

* Find three operands that add up to 20.
* The operands can have a value between -19 and 19.

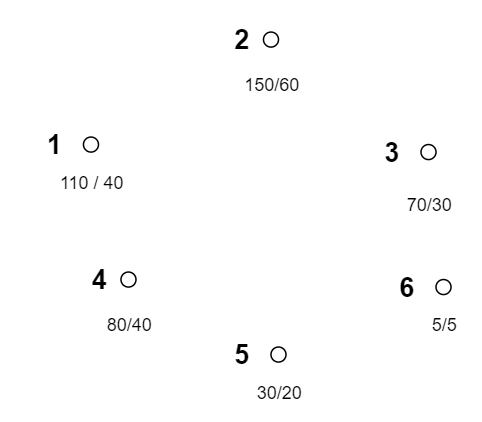
Examples:

15 + -3 + 2 = 14

1 + 6 + 11 = 18

1. **Metaheuristic:** Ant System (AS)
   1. Run only two (2) complete cycles of searching for an Ant System trying to solve the MKP instance. A cycle ends when each ant cannot add more items to its knapsacks and a new level of pheromone of each arc of the graph is calculated.
   2. Simulate an AS with m = 5 ants (not necessarily every ant should start in a different node) and show the selected value for each parameter of the method.
   3. You mush show each required pseudorandom number, each performed calculation, and the effect of each step of the method.

Graph representation



**Parameters:**

**N**: Heuristic for choosing next item = value/weight of item, when it fits on either of the sacks

**Alpha**: 1, **Beta**: 1

*Having the same values for alpha and beta, both the heuristic and the trial are equally important for the algorithm.*

**P** = 0.5 pheromone consumption

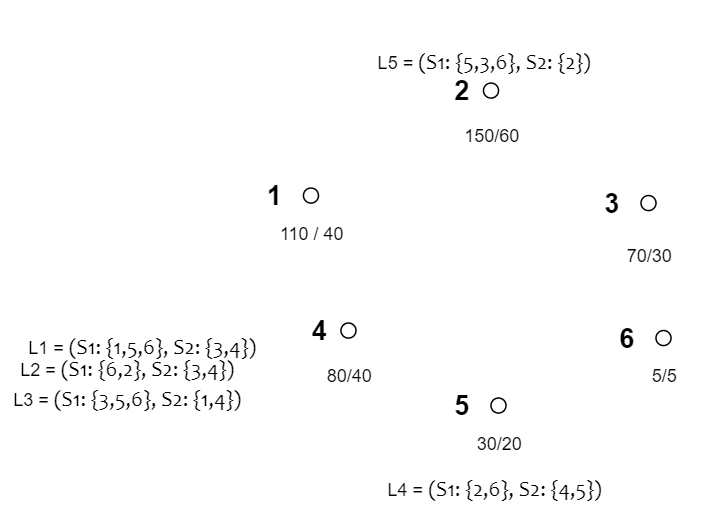
**Q** = total possible profit (445)

**Lk** = ant profit

**Ti** = Initial pheromone trial = 0.1

**∆τ** = Lk / Q

**Iteration 1**



L1 = (S1: {1,5,6}, S2: {3,4}) = weights (65, 70) = profit 295

**L2 = (S1: {6,2}, S2: {3,4}) = weights (65,70) = profit 305**

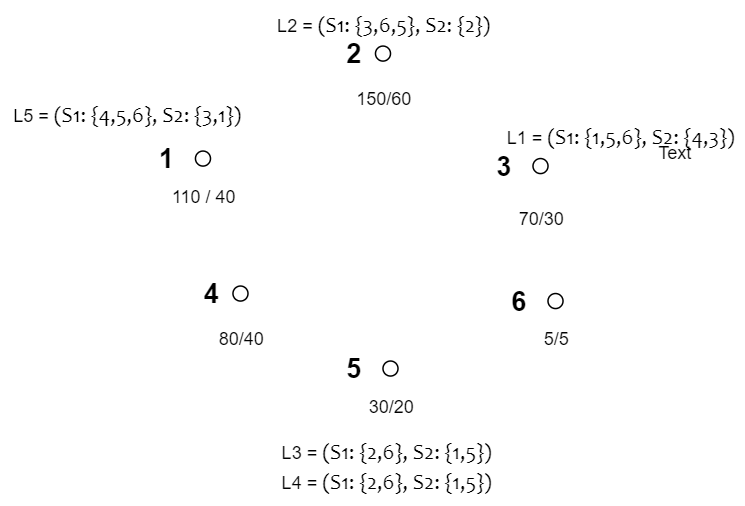
L3 = (S1: {3,5,6}, S2: {1,4}) = weights (55, 80) = profit 295

L4 = (S1: {2,6}, S2: {4,5}) = weights (65, 60) = profit 265

L5 = (S1: {5,3,6}, S2: {2}) = weights (55,60) = profit 255

**∆τ** total = 295 / 445 + 305/445 + 295/445 + 265/445 + 255/445 = 0.66 + 0.69 + 0.66 + 0.60 +0.57=3.18

**Iteration 2**



**L1 = (S1: {1,5,6}, S2: {3,4}) = weights (65, 70) = profit 295**

L2 = (S1: {3,6,5}, S2: {2}) = weights (55,60) = profit 255

L3 = (S1: {2,6}, S2: {1,5}) = weights (65, 70) = profit 295

L4 = (S1: {2,6}, S2: {1,5}) = weights (65, 70) = profit 295

L5 = (S1: {4,5,6}, S2: {1,3}) = weights (65,70) = profit 295

**∆τ** total = 295 / 445 + 255/445 + 295/445 + 295/445 + 295/445 = 0.66 + 0.57 + 0.66 + 0.66 +0.66=3.21

**Global Best:** **(S1: {6,2}, S2: {3,4}) = weights (65,70) = profit 305**

1. **Metaheuristic:** Bees Algorithm (AB)
   1. Run two (2) complete cycles of a Bees Algorithm trying to solve the SAP instance. A cycle ends when the remaining bees for random search (scout bees) are assigned.
   2. Simulate a BA with n = 7 bees and show the selected value for each remaining parameter of the method.
   3. You must show each required pseudorandom number, each performed calculation, and the effect of each step of the method.
2. **Metaheuristic:** Particle Swarm Optimization (PSO)
   1. Run two (2) complete cycles of a Particle Swarm Optimization system trying to solve the SAP instance. A cycle ends when each particle for random search has its new velocity and position computed.
   2. Simulate a PSO with n = 7 particles with circular neighborhoods of size 3 and a maximum velocity (VMAX) of 5.
   3. You must show each required pseudorandom number, each performed calculation, and the effect of each step of the method.

For the SAP presented to us we must deal with a problem size of 3 variables (x, y and z), these have a domain [-19, 19] in the set of real numbers. The domain of velocities will be [-5, 5] in the set of real numbers as well. The objective function we will try to minimize is . We start the algorithm by creating 7 particles with random positions and velocities in a 3-dimensional space:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Position |  |  |  |  |  |  |  |
| Cost | 30 | 54 | 14 | 35 | 40 | 8 | 4 |
| Velocity |  |  |  |  |  |  |  |
| Best position |  |  |  |  |  |  |  |
| Best cost | 30 | 54 | 14 | 35 | 40 | 8 | 4 |

Note that the best position/cost for each particle initially is trivially the random initialization. We will also identify the neighborhoods best particle for each particle using circular neighborhoods of size 3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Calculate | Min(  P7.cost, P1.cost, P2.cost) | Min(  P1.cost, P2.cost, P3.cost) | Min(  P2.cost, P3.cost, P4.cost) | Min(  P3.cost, P4.cost, P5.cost) | Min(  P4.cost, P5.cost, P6.cost) | Min(  P5.cost, P6.cost, P7.cost) | Min(  P6.cost, P7.cost, P1.cost) |
| Neighborhood best | P7 | P3 | P3 | P3 | P6 | P7 | P7 |

BEGIN ITERATION 1

First we update the velocities of each particle using the formula

Where *vi* is the velocity in a given iteration *i*, *c1* and *c2* are constant tuning parameters (we’ll set them to 2 for this exercise), *r1* and *r2* are random reals between 0 and 1, *X* is the current position vector of the particle in question, *b* is the best position vector of the particle in question and *B* is the best position vector of the best particle in the neighborhood of the particle in question (the maximum absolute velocity that each individual component of a velocity vector should be able to reach is 5 as specified in the problem):

Particle 1:

Cap velocity:

Particle 2:

Cap velocity:

Particle 3:

Particle 4:

Cap velocity:

Particle 5:

Cap velocity:

Particle 6:

Cap velocity:

Particle 7:

Now we update the position of each particle with the following formula:

Where *Xi* is the position vector in iteration *i* of the particle in question and *vi* is the velocity vector in iteration *i* of the particle in question (we just calculated these):

Particle 1:

Particle 2:

Particle 3:

Particle 4:

Particle 5:

Particle 6:

Particle 7:

We may now update our population of particles to what we have at the end of iteration 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Position |  |  |  |  |  |  |  |
| Cost | 21.5 | 51.8 | 12.3 | 34.7 | 35 | 7.5 | 3.9 |
| Velocity |  |  |  |  |  |  |  |
| Best position |  |  |  |  |  |  |  |
| Best cost | 21.5 | 51.8 | 12.3 | 34.7 | 35 | 7.5 | 3.9 |

Notice we have updated the best positions and best costs accordingly. It was the case that every particle moved to a better position compared to the initial. Even particle 7 which was the global best improved its position by 0.1 when computed by the objective function. We proceed with the neighborhoods calculation which didn’t change compared to the initialization:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Calculate | Min(  P7.cost, P1.cost, P2.cost) | Min(  P1.cost, P2.cost, P3.cost) | Min(  P2.cost, P3.cost, P4.cost) | Min(  P3.cost, P4.cost, P5.cost) | Min(  P4.cost, P5.cost, P6.cost) | Min(  P5.cost, P6.cost, P7.cost) | Min(  P6.cost, P7.cost, P1.cost) |
| Neighborhood best | P7 | P3 | P3 | P3 | P6 | P7 | P7 |

BEGIN ITERATION 2

Particle 1:

Cap velocity:

Particle 2:

Cap velocity:

Particle 3:

Particle 4:

Cap velocity:

Particle 5:

Cap velocity:

Particle 6:

Cap velocity:

Particle 7:

Now computing the new distances

Particle 1:

Particle 2:

Particle 3:

Particle 4:

Particle 5:

Particle 6:

Particle 7:

We may now update our population of particles to what we have at the end of iteration 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Position |  |  |  |  |  |  |  |
| Cost | 6 | 31.2 | 11.4 | 45.1 | 24.1 | 2.3 | 3.8 |
| Velocity |  |  |  |  |  |  |  |
| Best position |  |  |  |  |  |  |  |
| Best cost | 6 | 31.2 | 11.4 | 34.7 | 24.1 | 2.3 | 3.8 |

We notice that this time 1 particle (P4) didn’t improve its cost compared to the last iteration, but every other particle still did. P6 is the new global best particle with a cost of 2.6 in spite of P7 improving from 3.9 in the last iteration to 3.8.