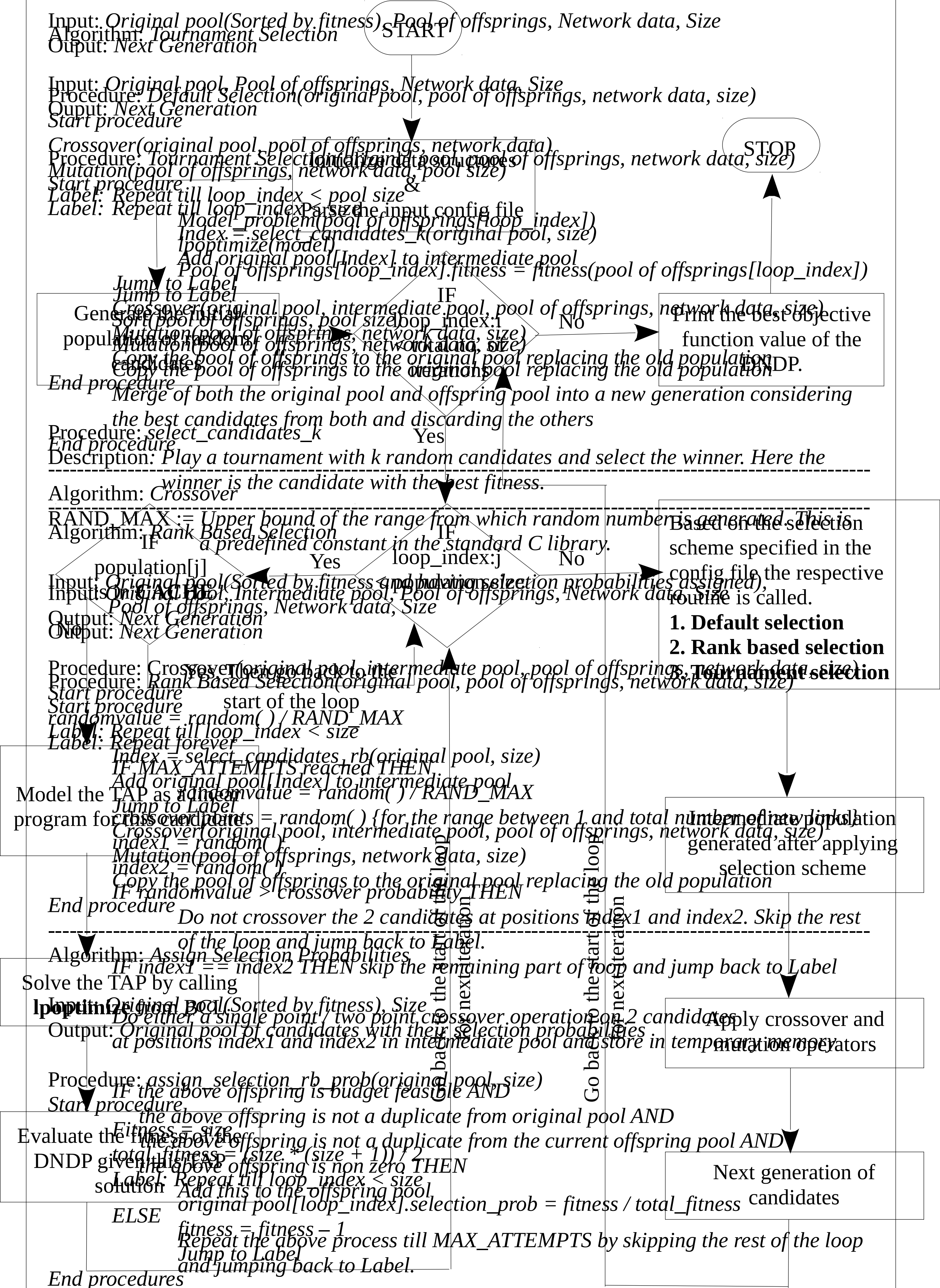
START
Initialize data sources
&
Parse the input config file
Generate the initial population of random candidates
IF
loop_index:i
< total no. of iterations
No
Yes
Print the best objective function value of the DNDP.
STOP
IF
loop_index:j
< population size
Yes
No
IF
population[j]
is ACTIVE
No
Yes. Then go back to the start of the loop
Model the TAP as a linear program for this candidate
Solve the TAP by calling **lpoptimize** from BCL
Evaluate the fitness of the DNDP given this TAP solution
Based on the selection scheme specified in the config file the respective routine is called.
1. Default selection
2. Rank based selection
3. Tournament selection
Intermediate population generated after applying selection scheme
Apply crossover and mutation operators
Next generation of candidates
Go back to the start of the loop for next iteration
Go back to the start of the loop for next iteration

```
Input: Original pool, Pool of offsprings, Network data, Size
Ouput: Next Generation
Procedure: Default Selection(original pool, pool of offsprings, network data, size)
Start procedure
Crossover(original pool, pool of offsprings, network data)
Mutation(pool of offsprings, network data, pool size)
Label: Repeat till loop_index < pool size
        Model_problem(pool of offsprings[loop_index])
        lpoptimize(model)
        Pool of offsprings[loop_index].fitness = fitness(pool of offsprings[loop_index])
    Jump to Label
    Sort(pool of offsprings, pool size)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
    Merge of both the original pool and offspring pool into a new generation considering
    the best candidates from both and discarding the others
End procedure
```

```
Algorithm: Tournament Selection
Input: Original pool(Sorted by fitness), Pool of offsprings, Network data, Size
Ouput: Next Generation
Procedure: Tournament Selection(original pool, pool of offsprings, network data, size)
Start procedure
Label: Repeat till loop_index < size
        Index = select_candiddtes_k(original pool, size)
        Add original pool[Index] to intermediate pool
    Jump to Label
    Crossover(original pool, intermediate pool, pool of offsprings, network data, size)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
End procedure
Procedure: select_candidates_k
Description: Play a tournament with k random candidates and select the winner. Here the
        winner is the candidate with the best fitness.
----------------------------------------------------------------------------------------------------
```

```
Algorithm: Crossover
RAND_MAX := Upper bound of the range from which random number is generated. This is
        a predefined constant in the standard C library.
Input: Original pool, Intermediate pool, Pool of offsprings, Network data, Size
Output: Next Generation
Procedure: Crossover(original pool, intermediate pool, pool of offsprings, network data, size)
Start procedure
Label: Repeat forever
    IF MAX_ATTEMPTS reached THEN
        randomvalue = random( ) / RAND_MAX
    crossover points = random( ) {for the range between 1 and total number of new links}
    index1 = random( )
    index2 = random( )
    IF randomvalue > crossover probability THEN
        Do not crossover the 2 candidates at positions index1 and index2. Skip the rest
        of the loop and jump back to Label.
    IF index1 == index2 THEN skip the remaining part of loop and jump back to Label
    Do either a single point / two point crossover operation on 2 candidates
    at positions index1 and index2 in intermediate pool and store in temporary memory
    IF the above offspring is budget feasible AND
        the above offspring is not a duplicate from original pool AND
        the above offspring is not a duplicate from the current offspring pool AND
        the above offspring is non zero THEN
        Add this to the offspring pool
    ELSE
        Repeat the above process till MAX_ATTEMPTS by skipping the rest of the loop
        and jumping back to Label.
End procedures
```

```
----------------------------------------------------------------------------------------------------
Algorithm: Rank Based Selection
Input: Original pool(Sorted by fitness and having selection probabilities assigned),
        Pool of offsprings, Network data, Size
Output: Next Generation
Procedure: Rank Based Selection(original pool, pool of offsprings, network data, size)
Start procedure
randomvalue = random( ) / RAND_MAX
Label: Repeat till loop_index < size
        Index = select_candidates_rb(original pool, size)
        Add original pool[Index] to intermediate pool
    Jump to Label
    Crossover(original pool, intermediate pool, pool of offsprings, network data, size)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
End procedure
----------------------------------------------------------------------------------------------------
Algorithm: Assign Selection Probabilities
Input: Original pool(Sorted by fitness), Size
Output: Original pool of candidates with their selection probabilities
Procedure: assign_selection_rb_prob(original pool, size)
Start procedure
    Fitness = size
    total_fitness = (size * (size + 1)) / 2
    Label: Repeat till loop_index < size
        original pool[loop_index].selection_prob = fitness / total_fitness
        fitness = fitness – 1
        Jump to Label
```