```
1:  procedure communityDetection(k, data, alpha, delta, epsilon)

    /* INITIALIZE DATA STRUCTURES ***********************************************************/

    /* COMMENTS:
     * Line 2: Read in data and implement graph as an adjacency list with n raws (= num of
     nodes).
     * Line 3: Create the particle objects and save them in a vector P, dim(P) = k x 1.
     * Lines 4, 5: Disperse all k particles randomly in graph G. Calculate initial particle
     energy levels (attribute in particle struct).
     * Line 6: Initialize particle state (attribute in particle struct). Initially all
     particles have state ACTIVE (vs EXHAUSTED).
     * Line 7: Initialize matrix N, containing the number of times each particle visits EACH
     node in G. dim(N) = n x k.
     * Line 8: Compute matrix NBar, containing the relative visit frequencies of all particles
     to all nodes in G. dim(NBar) = n x k.
     * Line 9: Compute Prand, the matrix containing the probability that a particle p,
     occupying node i at time t, would move to node j at time t+1.
     * Prand is Markovian, proportional to the edge weight linking nodes i and j. It is
     time-invariant and thus computed only once.  */
2:      G <- buildGraph(data)
3:      P <- initParticles(k)
4:      initParticleDisperse(G)
5:      calcInitE(P, k)
6:      initParticleState(P,k)
7:      N <- initMatrixN(G,P)
8:      NBar <- compMatrixNBar(N)
9:      Prand <- compPrand(G)

    /* RUN DETECTION ********************************************/

    /* COMMENTS:
     * Lines 10-24: From time t=1 until a fixed point is reached, move each particle on G as
     follows -
     * Lines 13, 14: If the particle is ACTIVE, compute the preferential movement probabilities
                    for all neighbors of P[k].current_node. That is, calculate the preferential
                    probability that particle k will move to node j, a neighbor of node i, k's
                    current residence.
                    The probabilities are stored in a vector of size G[i].number_neighbors,
                    where i = P[k].current_node.
                    The relative visit frequencies from matrix NBar are used in the computation.
                    dim(pref_prob) = 1 x number_neighbors.

     * Line 15: Compute the transition vector. That is,
                tran_vector[i] = (1-alpha)*rand_prob[i] +alpha*pref_prob[i],
                where i is the i-th neighbor of P[k].current_node and rand_prob[i] is
                the i-th entry in raw P[k].current_node in Prand.
     * Line 16: Move particle k to a new node based on tran_vector.
     * Line 17: Update the energy level of particle k (+-delta).
     * Lines 18, 19: If particle k is EXHAUSTED, reanimate it by teleporting it
                to a node it dominates (selected based on weighted rand distrib.)
     * Line 21: Update matrix N to increment the number of node visits where appropriate.
     * Line 22: Update matrixNBar to reflect the change in visit frequencies.
```

```
*/

    t <- 1
    do
        for k = 1 to |P| do
            if(P[k].state == ACTIVE)
                pref_prob = computePrefProb(P[k])
                tran_vector = compTransitionVector()
                selectNextNode(P[k], tran_vector, num_neighbors)
                updateE(P[k])
            else if(P[k].state == EXHAUSTED)
                particleRean(P[k])
        end for

        updateMatrixN()
        compMatrixNBar(N)
        t <- t+1

    while inf_norm(NBar(t)- NBar(t-1)) < epsilon        //compute a quasi-fixed-point
end procedure
```