1. The tasks are all implemented. Please see /doc/readme.txt for more information and /output/ for the learned models stored as UAI files and results.xlsx|pdf for the log likelihood differences.
   1. Task 1 is implemented as BN\_MLE\_FOD and reported with the same name. Please see /doc/readme.txt for more information.
   2. Task 2 is implemented as BN\_EM\_POD and reported with the same name. Please see /doc/readme.txt for more information. Note that the proposal distribution is *randomly generated* and so the lld values reported reflect that. Each reported value is after 20 iterations.
   3. Task 3 is implemented as BS\_CL\_FOD and reported with the same name. Please see /doc/readme.txt for more information.
2. Der
   1. Gibbs sampling is a special case of Metropolis-Hastings. In the case of the proposal distribution Q is the uniform distribution. If Q is the uniform, then the M-H acceptance probability will always be 1. Thus the transition will always be accepted and the state will always change. This is the same as Gibbs sampling.
   2. This answer is yes and no. Consider two cases: the Markov network is not a chain or the Markov network is a chain.
      1. If the Markov network is a chain, then having probabilities of 0 in the functions means the network will not converge, as some states may never be reached. If this happens, the network is not ergodic.
      2. If the Markov network is not a chain, then having probabilities of 0 in the functions does not necessarily mean the network will not converge. States may yet be reachable through other means. Consider the network A-B-C-A (K3). If A can never transition to B directly, there may still be a way for A to transition to B through C.
3. The network looks like this:
   1. The various complexities are discussed below.
      1. VE\_MPE: O(niexp(w)). The biggest problem here is the ordering. A poor ordering (in this case, not eliminating C first) would make line 6 atrocious. However, depending on the ordering, line 6 does have a terrible O complexity. In fact, if the ordering is the same as an MAP ordering (like in the next part) the complexity is exactly the same. Each line is analyzed below:
         1. O(1) – no prunes happen.
         2. O(1) – given.
         3. O(n) – assuming order can be computed in linear time.
         4. O(1) – assuming evidence is already instantiated.
         5. O(m) – m is the number of variables in the network.
         6. O(i\*exp(w)) where w is max(number of variables in each multiplied function) and I is the number of non evidence variables.
         7. O(n) – max or sum out is linear.
         8. O(1) – just replacing pointers.
         9. n/a
         10. O(1) – a single return.
      2. VE\_MAP: O(niexp(w)). The biggest problem here is the ordering. Since this is a MAP ordering, we have to sum out C in all n of the A CPTs. This causes a horrendous runtime complexity. The analysis of this algorithm is similar to that of the MPE algorithm.
      3. With evidence on C, the complexities change to: O(nd) where n is the number of A variables and d is max(domain sizes for each A). MPE and MAP have the same complexity in this case because the edges from C to A\_i to A\_n can effectively be ignored, and all the A\_i variables are independent given C. So a single sum out is necessary on each A\_i CPT, then a multiplication of all the trivial functions left over.
4. True. This approach is essentially the naïve way to compute MAP by computing the MPE of every combination of variable values and taking the max. This approach still works even if X is part of M.
5. If all MAP variables are contained in a single cluster, then without loss of generality, we can pick that cluster as the root of a jointree propagation algorithm. We compute O(n) messages and send them to the root, and each message computation takes O(w) time because it is multiplying CPTs. Observe the following example:  
   3-3.png