Part II: Multi-robot Path Planning for a Given Itinerary

TBA

May 25, 2004

1 Problem A of part II

1.1 Problem definition

Imagine a situation where we have multiple robots to travel an itinerary. We want to plan a set of closed paths[1], or circles in short, each for every robot, such that the union of the paths will cover the iterary and at the same time, the total distance, the sum of all the travelling distances of all the robots, is minimized.

In the "Prolem A" of this situation, we assume that both the number of robots and their initial start points/nodes, and hence their end points/nodes, in the graph are fixed.

Here we give the formal mathematical form of this problem.

A bi-directed graph G is defined as $G = (V, E)$ in which $V = \{v_i, 1 \leq i \leq n\}$ is the set of all nodes in the graph and $E = \{e_{i,j}, 1 \leq i \neq j \leq n\}$ is the set of all edges. Each $e_{i,j} =< v_i, v_j >$ represents an arc that the robot can travel either from node v_i to v_j or from node v_j to v_i. There is also a cost function $cost(e_{i,j}) = c_{i,j}$ defined on each edge $e_{i,j}$ as the distance travelled by the robot of the $e_{i,j}$ between nodes v_i and v_j.

For a given graph G, we also define the set of m itinerary nodes, $\mathcal{I} = \{\mathcal{I}_{\updownarrow}, \infty \leq \updownarrow \leq \Updownarrow\}$, as the set of the nodes that have to be travelled by at least one robot at least once. Then a set of robots, $R = \{r_k, 1 \leq k \leq K\}$, (Robot r_k starts and ends at node $v^k_{p_1}$.) are given to travel the itinerary.

Given these settings, the Problem A addressed here is to find a set of closed paths or circles, p_k, $1 \leq k \leq K$, each p_k is assigned to robot r_k, where $p_k = (v_{p^k_1}, v_{p^k_2}, \ldots v_{p^k_{end_k}})$, and $v_{p^k_{end_k}} = v_{p^k_1}$. (In the above, we assume that all the nodes in the path are in the graph, i.e., $v^k_{p_1} v_{p^k_2}, \ldots v_{p^k_{end_k}} \in V$.) To make sure that each path p_k for robot r_k is also connected by the edges in E, we have the following constraints, $< v_{p^k_1}, v_{p^k_2} >\in E$, $< v_{p^k_2}, v_{p^k_3} >\in E$, $\ldots$, $< v_{p^k_{end_k-1}}, v_{p^k_{end_k}} >\in E$. And the optimal criteria is given as:

$$\arg \min_{p_k, 1 \leq k \leq K} \sum_k cost(< v_{p^k_1}, v_{p^k_2} >) + cost(< v_{p^k_2}, v_{p^k_3} >) + \cdots + cost(< v_{p^k_{end_k-1}}, v_{p^k_1} >)$$

such that the union of all the nodes in the paths is the superset of V,

$$V \subseteq \bigcup_{p_k, 1 \leq k \leq K} (v_{p^k_1} \cup v_{p^k_2} \cup \cdots \cup v_{p^k_{end_k-1}})$$

1.2 Solution

(Please refer to the first part of our report for some discussion on related works on Travel Salesman Problem (TSP).)

[1] "Closed" here means that the paths start and end at the same node.

1.2.1 Transformation from problem A to TSP with r robots

Based on the observation that if we make the number of robots to be 1 and all the nodes in V to be the itinerary nodes, our problem A is reduced to the "TSP with multiple visits (TSPM)". Thus our problem A is at least as hard as the TSPM. Inspired by this, our solution to problem A is first to transform it to the TSPM with r robots.

The transformation is done by constructing a different graph $\tilde{G} = (\tilde{V}, \tilde{E})$ and the cost function defined on $\tilde{E}$), in which $\tilde{V}$ is the union of the itinerary nodes and robot starting nodes, $\tilde{V} = I$, and the edge $\tilde{e}_{i,j} =< \tilde{v}_i, \tilde{v}_j > \in \tilde{E}$ exists if there exists a shortest path between them in the original graph G, and the cost of this edge is the cost of this shortest path. We call the new graph $\tilde{G}$ the shortest path graph.

The algorithm of this tranformation is as follows,

Algorithm: Transformation from problem A to TSP with r robots

$\tilde{V} = I$ *% such that* $\tilde{v}_i = I_i, 1 \leq i \leq m$
$\tilde{E} = \phi$
for $1 \leq i \neq j \leq m$
    % using the shortest path algorithm
    Compute the shortest path in G between I_i *and* I_j *with distance* $length(I_i, I_j)$
    if $length(I_i, I_j) < \infty$
        $\tilde{E} = \tilde{E} \cup \tilde{e}_{i,j}$
        $\tilde{c}_{i,j} = cost(< \tilde{v}_i, \tilde{v}_j >) = length(I_i, I_j)$
$\tilde{G} = (\tilde{V}, \tilde{E})$

The original problem now becomes the one to find circles, each for every robot that starts and ends at its specific starting node, to travel all the nodes in $\tilde{V}$ exactly once. We call this problem "Multi-robot TSP". After solving this "Multi-robot TSP", we can easily transform this solution, a sequence of itinerary nodes, back to the solution to the original prblem. This is done by adding between two consecutive nodes in the solution all the edges that the robot has to travel to get this shortest distance.

In the following, we are going to discuss how to solve this "Multi-robot TSP".

1.2.2 Solution to "Multi-robot TSP"

Again, we should re-emphasize here that the problem now is for all the robots to travel exactly once the all the nodes in the shortest path graph, $\tilde{G}$.

We have done so far basically three solutions, one based on heuristics, one based on linear programming (LP), and one based on the transformation of the problem to TSP of a different graph.

1. Jiaping Algorithm

The first solution we got was contributed by Jiaping. We call it "Jiaping's Algorithm". "Jiaping's Algorithm" was based on the heuristic of one robot's itinerary travelling solution. Intuitively, if one robot can travel all the itinerary nodes by following the path p' in reasonable time, a partition of p' into K parts, each for every robot to travel, would also be reasonably short intuitively.

Based on this, Jiaping's algorithm is as follows. First, solve the TSP on $\tilde{G}$. We denote the solution by p' [2]. Now as the third step, we partition p' into r pieces, each assigned to every robot for it to travel. The partition is done by cutting the longest edge between two consecutive nodes in p'. The itinerary nodes connected to robot r_k after cutting is assigned to the set of nodes designated to r_k to travel. Then, for each element of the partition, we solve the TSP using methods in Part I of our report.

This longest cost cutting heuristic may not give us the optimal solution as we can see in the simulation results.

2. Solution based on Integar Linear Programming (IP)

[2] As we can easily see that the choice of r' to start with does not affect the result, i.e., we will have the same p' no matter which robot is used.

Another possible solution is to formulate the "Multi-robot TSP" into the standard Integer Linear Programming (IP). In the IP formulation, we should define a set of binary variables $\tilde{x}^k_{i,j}$, each $< i, j >$ runs for all the edges $\tilde{G}$ and k runs for all the robots, to denote whether the edge $\tilde{e}_{i,j}$ is travelled by robot r_k. If $\tilde{x}^k_{i,j} = 1$, the robot r_k will go through the edge $\tilde{e}_{i,j}$; $\tilde{x}^k_{i,j} = 0$, the robot will not go through the edge $\tilde{e}_{i,j}$. Then, the IP for this problem is given by the following formulation.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} \tilde{x}^k_{i,j}, \ \textit{where } \tilde{x}^k_{ij} \in \{0,\ 1\} \tag{1}$$

$\tilde{x}^k_{i,j}$ means whether $\tilde{e}_{i,j}$ is in the k^{th} robot's path. (1 means to travel it; 0 means not.)

$$y^k_i \in \{0,\ 1\} \tag{2}$$

This indicates whether the k^{th} robot travel i^{th} node. And this could be thought of as a partition of the nodes to each robot for it to travel.

$$\textstyle\sum_k y^k_i \geq 1 \text{ for each i}$$

$$\textstyle\sum_k y^k_i \leq K,\ K \text{ is the number of robots}$$

This guarantees that every node is traveled at least once by a robot.

$$\tilde{x^k_{ij}} \leq y^k_i \text{ for each j}$$

This guarantees that if the k^{th} robot visits i^{th} node, indicated by $y^k_i = 1$, it travels along some adjacent edge to this node.

Then we have to add subtour elimination constraints given as follows:

$u(i,k) - u(j,k) + N \cdot \tilde{x^k_{ij}} \leq N - 1$, *for* $i, j \neq$ *robot starting or ending nodes.*

The u's in the above equation are variables.

3. Transformation from "Multi-robot TSP" to TSP

This approach is proposed by Pengpeng, based on the following observation. If we break the circles of the solution to Multi-robot TSP, and chain them together to make a single circle such that the end of the previous piece is connected to the start of the next piece by a zero cost edge, the total cost of the Multi-robot TSP solution will be the same as this single circle.

And the algorithm is as follows,

Algorithm: Transformation from Multi-robot TSP to TSP

$\tilde{V}' = \tilde{V}$
$\tilde{E}' = \tilde{E}$
% Remember the robots' starting nodes are called $\tilde{v}^k_1$, $1 \leq k \leq K$
Make a copy of $G' \cap R$
Name the copy $RV = rv_k, 1 \leq k \leq K$ *% the virtual node set*
% Add virtual nodes to the original graph
$\tilde{V}' = \tilde{V} \cup RV$
% Connect virtual nodes to its corresponding $\tilde{v}^k_1$ *nodes by 0-cost edge*
Make edges $< \tilde{v}^k_1, rv_k >$, $1 \leq k \leq K$
$\tilde{E}' = \tilde{E}' \cup (\cup_k < \tilde{v}^k_1, rv_k >)$
$cost(< \tilde{v}^k_1, rv_k >) = 0,\ 1 \leq k \leq K$
% Connect virtual nodes to the nodes that is connected to their corresponding nodes in $\tilde{G}$
if $< \tilde{v}^k_1, \tilde{v}_{i'} > \in \tilde{E}$
    Make edge $< rv^k_1, \tilde{v}_{i'} >$
    $\tilde{E}' = \tilde{E}' \cup < rv^k_1, \tilde{v}_{i'} >$

$cost(< rv_1^k, \tilde{v}_{i'} >) = 0$

% Make the complete graph of robot starting nodes with zero-cost edges
for $1 \leq k \neq k' \leq K$
Make edge $< \tilde{v}_1^k, \tilde{v}_1^{k'} >$
$\tilde{E}' = \tilde{E}' \cup < \tilde{v}_1^k, \tilde{v}_1^{k'} >$
$cost(< \tilde{v}_1^k, \tilde{v}_1^{k'} >) = 0$

% Make the complete graph of robot starting nodes with zero-cost edges
for $1 \leq k \neq k' \leq K$
Make edge $< rv_1^k, rv_1^{k'} >$
$\tilde{E}' = \tilde{E}' \cup < rv_1^k, rv_1^{k'} >$
$cost(< rv_1^k, rv_1^{k'} >) = 0$

$\tilde{G}' = (\tilde{V}', \tilde{E}')$

Then we need to solve the TSP on $\tilde{G}'$ and transform the solution p' back to the initial problem. The algorithm is as follows. First, we collapse the virtual nodes and their corresponding robot starting nodes together. Then there will be one circle at each robot starting node (Remember this is a TSP so the sum of the numbers of visits at each robot starting node and its virtual node is exactly two.), if they are not connected by a 0-cost edge. If a robot node and its virtual node are actually connected by a 0-cost edge, our algorithm will make this robot stay there and not travel.

So we only consider how to provide paths for those travelling robots. First, these robots are scheduled to travel its circle. Then, for two adjacent robot starting nodes on the collapsed version of p', we can pick either of the robots to tavel the piece on collapsed p' between them and then go back by the shortest path its starting nodes.

After discussions, we realized that the paths we got from this are not necessarily the optimal solution to Multi-robot TSP. However, in some cases where there is no other node between two adjacent robot starting nodes on the collapsed p', the solution given above is optimal. The reason is as follows. For the optimal solution of Multi-robot TSP, we can make virtual nodes and chain them by 0-cost edges as mentioned before. The circle made then is a solution to the Multi-robot TSP. And at the same time, the transformation mentioned above can make the optimal solution to the Multi-robot TSP to a solution to TSP. So it must be optimal to TSP too.

But for cases where there are other nodes between two adjacent robot starting nodes on the collapsed p', the transformed solution we get is not optimal. However, we can say something on the solution. That is, the solution is bounded by the cost of the optimal solution plus the sum of the shortest distances between robot starting nodes. This can be intuitively explained by the fact that the sum of the cost of the circles is less than or equal to the cost of the optimal solution. And for the robot to be able to travel back to its starting node, the additional cost is at most the sum of the shortest distances between robot starting nodes.

1.2.3 Heuristic solution to problem A

TSP is a NP-complete problem and well accepted that exponential computational time is needed to get an optimal solution (unless "NP=P"). This is why for $n > 100$ we would compromise for a sub-optimal solution for the sake of computation. Here we are going to talk about how to solve the problem by heuristics. The issue here is to get a good solution reasonably fast, although the solution may not be optimal.

The strategy we use is called the "Heuristic Partition". Again, first we compute the shortest path graph $\tilde{G}$. Then assign each itinerary node to its nearest robot. After the assignments of all the nodes, we obtain a partition of the itinerary, one element of the partition for each robot. We can then solve the problem by solving the single robot as discussed in Part I of our report.

Formally, the algorithm is as follows. (In the following, we denote the robot starting nodes as $\tilde{v}_1^k$, to be consistent with our former nonmenclature.)

Algorithm: Heuristic Partition

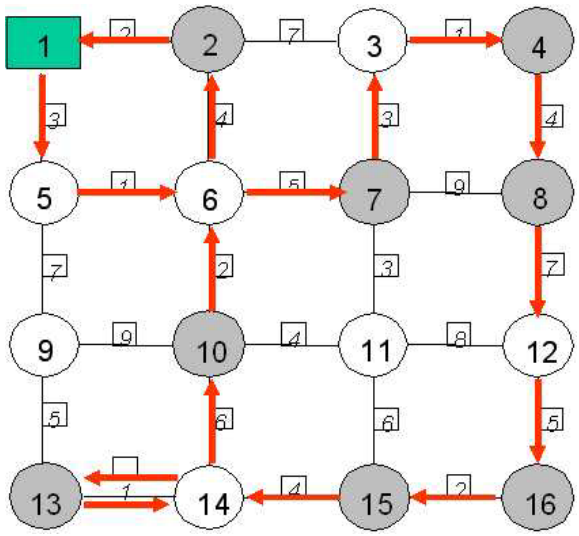

Figure 1: Single Robot planning result using algorithm in Part I. The robot starts at node 1, denoted by square, travels as directed by the arrows, and ends at node 1.

% The partition sets
$Part_k = \phi, 1 \leq k \leq K$
Compute the shortest path graph $G = (V, E)$ *with costs* $c_{i,j}$
For each $\tilde{v}_i \in \tilde{V}$
  % Choose the robot $r_{k'}$ *such that cost*$(< v_1^{k'}, v_i >)$ *is minimal*
  $r_{k'} = \arg\min_{r_{k'}} cost(< v_1^{k'}, v_i >)$
  $Part_{k'} = Part_{k'} \cup r_{k'}$
% Solve the one-robot planning problem for each partition
for $k = 1 \ldots K$
  Solve one-robot planning problem for $Part_k$

In solving the one-robot planning problem for each partition, we can also use some heuristics as mentioned in Part I of our report.

If we use the exact solver in Part I to solve the one-robot planning problem, we know that our solution is at least better than to use a single robot to travel all the itinerary nodes. However, the overall performance will depend on the quality of the partition and we cannot say anything on this. One more comment we would like to mention here is that this heuristic can be used to partition not only the itinerary nodes but the other nodes in the original graph. This will be especially useful for cases where we do not know what the itinerary nodes are a priori. The application situation would be for example in exploration, we get the map in advance but the tasks of where to explore are given different from time to time. This heuristic partition can then give us the same partition for different tasks and save a lot of computational time.

1.3 Simulation results

To test the efficiency of the algorithms, we implemented them in Matlab on the problems as in Figure

Also, the single robot result is shown in Figure 1 for comparison with the multiple robot planning cases.

The two problems above are based on the same graph with the same costs. However, the second one is for three robots starting at nodes 1, 4 and 13. (The itinerary nodes are colored grey.)

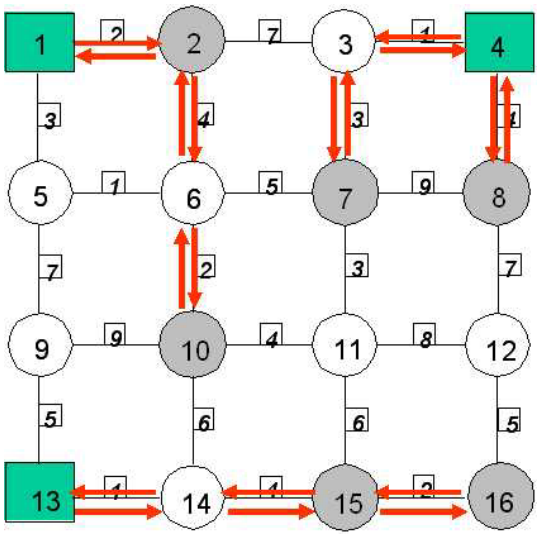

Figure 2: The result given by algorithms based on heuristic partition. Three robots start at specified nodes, denoted by squares, travel as directed, and end at the their initial positions.

It is interesting to see that both of the heuristic partition based algorithms gave the same result as in Figure 2.

The IP formulation implemented by the Matlab optimization toolbox gives the optimal solution as in Figure 3. It is interesting to see that the optimal solution in this case has the robot at node 13 staying, while the robots starting nodes 1 and 4 travel as in the figure.

2 Problem B of part II: fixed number of robots with choice of their placement

2.1 Uniformly distributed version

Here we assume that in addition to the model considered in Part II A we can choose the positions of the robots, based on the apriori information we have about the network. This situation is realistic and may arise for example in lifesaving operations, where number of automated devises usually search an area in order to minimize and guide the subsequent human effort - fires, earthquakes, avalanches, etc.

Therefore, we want to choose k nodes in the network of, as initial positions of the k robots, so that this will give us an opportunity to efficiently carry out searches on any given itinerary. Thus, we do not have any apriori information about the itinerary. One natural way to represent this problem is the so-called *k-median problem*. Given n points $p_1, p_2, \ldots, p_n$, we want to choose k of them as centers. Then we form clusters, assigning each of the remaining n-k points to the center that is closest to it. We want to choose the k centers so that the total distance from the points to the k cluster centres is minimized.

If p_{ij} stands for the j^{th} point assigned to the i-th center our goal is to

$$minimize \sum_{i=1}^{k} \sum_{i^{th}\ cluster} d(p_i, p_{ij})$$

When $k \geq 3$, the problem is NP-complete. We will use the approximation algorithm described in [1] The algorithm uses 3-stage optimization based on LP to achieve $6\frac{2}{3}$-approximation of the optimal, i.e. the k cluster centers selected by this algorithm guarantee total distance of no more than $6\frac{2}{3}$ times the optimal.

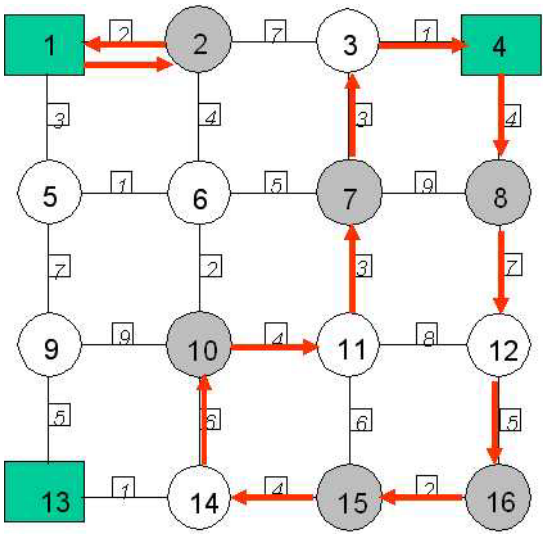

Figure 3: The optimal solution given by IP formulation.

The algorithm runs in polynomial time, dominated by the time necessary to solve the LP instances within. However, we cannot use this algorithm straightforwardly to solve our problem. In our network the distances between pair of locations might not be known, as generally there will be multiple paths connecting them, each path consisting of more than one edge. It is also possible that there will be no path between a pair of locations. Thus, we have to compute shortest paths between all pairs of nodes in our network and set the length of this shortest path as distance between these nodes for the purposes of the algorithm [1]. In addition we have to set the lengths of the non-existing paths to a sufficiently large number, say n times the length of the longest distance in the network. Now we are ready to run the algorithm and we will obtain the k cluster centers. At this point the problem is identical to the one of Part II A. Given an itinerary, we have our network subdivided in k clusters, we will identify the itinerary nodes belonging to each clusters, the robots will visit their respective parts of the itinerary based on shortest path computations done according to the single robot approach described in Part I.

Formally, the algorithm can be described as follows:

1. Find the shortest path between all pairs of nodes (p_i, p_j) in the network as described in Part I. Denote these lengths by $\tilde{c}_{ij}$. If no path exists, then we set $\tilde{c}_{ij} MaxD$, where $MaxD$ is a sufficiently large number.

2. Run the k-median algorithm on the initial network. The algorithm outputs a set of centres $(p_1, p_2, \ldots, p_k)$ and for each node in $(p_{k+1}, p_{k+2}, \ldots, p_n)$ the assignment to its respective centre.

3. Place the k robots at the centres $(p_1, p_2, \ldots, p_k)$ computed in step 2.

4. Given an itinerary $I = (I_1, I_2, \ldots, I_m)$, subdivide into k disjoint itineraries $I = \tilde{I}_1 \cup \tilde{I}_2 \cup \cdots \cup \tilde{I}_k$, s.t. $\tilde{I}_i \cap \tilde{I}_j = \phi$ for $i \neq j$. The nodes in $\tilde{I}_i$ belong to the cluster of the i-th robot.

5. Run the algorithm of Part I for robot i for the itinerary $\tilde{I}_i$.

2.2 Weighted distances

In the formulation of the problem in the previous section we assumed that each node is equally probable to become an itinerary node. In the real world we can imagine situations when we apriori know that some locations are more probable as itinerary locations than others. For example, police patrol checks in a city. Some streets, neighbourhoods or hot spots are checked more often, i.e. they are more probable to be on the itinerary. The way to incorporate this into our model is to define a probability distribution over the nodes of the network. Equivalently, we can assume that this distribution is normalized and each network location has a weight w_i assigned proportional to the initial probability for this node being an

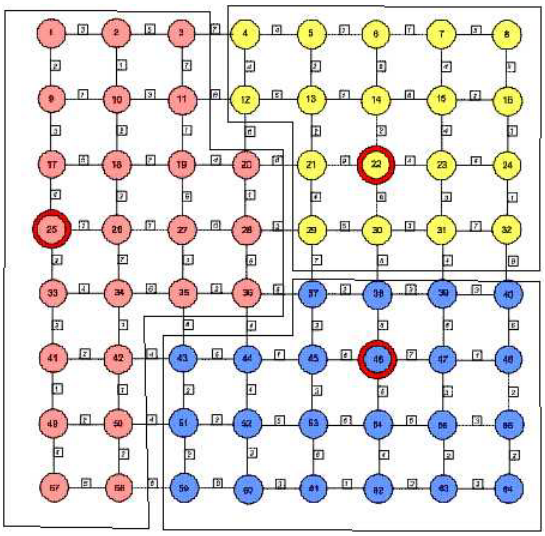

Figure 4: Clustering results for the unweighted network given by k-median algorithm. In the figure, all the nodes are divided into three clusters, as specified by the bounding boxes. (For the colored version, the clusters are denoted by "pink", "yellow" and "blue".) Each cluster is assigned to one robot starting at the node circled.

itinerary node. We want to place the robots so that the itinerary is carried out efficiently. Again, this can be done by clustering. In the literature this is known as the *k-median problem* with center costs. The algorithm from [1] can be easily modified to achieve a factor of 10 approximation of the optimal total distance. As one might expect, depending on the weights, the centers might not correspond to ones chosen in the unweighted k-median problem. As it is clear from this discussion, the only step that will be affected by this is step 2. Of course, we need some preprocessing, namely transforming the probability distribution into weight assignment. This can be done in either step 1 (as it does not affect shortest path computations) or in a separate step, before running the algorithm for the k-median problem with center costs.

2.3 Simulation Result

We implemented the *k-median* algorithm, both unweighted version and weighted one, in Matlab. Figures 4, 5 show the clustering results. It is interesting to note that the clustering results for these two networks that only differ by the weight of node 1 are the same, while the first robot starting position differ, one in node 25 and the other in 18.

3 Problem C of part II: unknown number of robots

In Problem C, we are considering to decide the minimal number of robots needed to travel the itinerary such that the total distance travelled by the robots are bounded by the value d. The strategy we are proposing is to first consider the Problem B with i robot. (The value of i starts from 1.) After solving this problem, we denote the total distance travelled corresponding to the solution, d_i. If d_i is less than d, we know the number of robots needed is i then. Otherwise, we consider Problem with $i + 1$ robots.

Note that this algorithm will not run for ever, because if we assign the same number of robots as the number of itinerary nodes, then we can simply put them on each node and let them not move at all. The total distance travelled is 0 in this case.

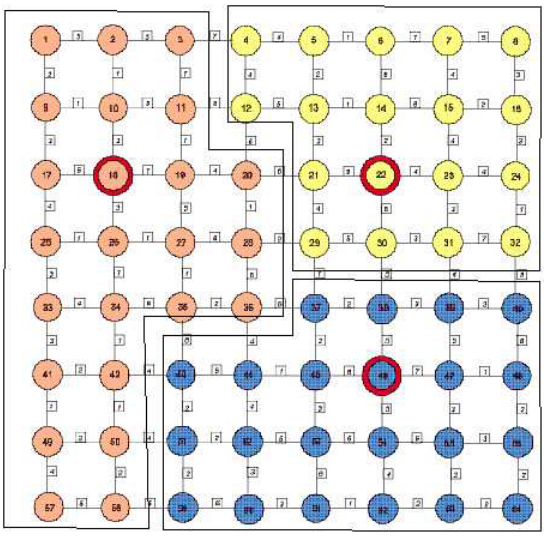

Figure 5: Clustering results for the weighted network given by k-median algorithm. The weights all nodes are the same except that of node 1, which is as 10 times heavy as the other. In the figure, all the nodes are divided into three clusters, as specified by the bounding boxes. (For the colored version, the clusters are denoted by "pink", "yellow" and "blue".) Each cluster is assigned to one robot starting at the node circled.

References

[1] M. Charikar, S. Gupta, E. Tardos, D. Shmoys, "A Constant-factor Approximation Algorithm for the k-median Problem", Proceedings of the ACM Symposium on Theory of Computing (STOC '99), Atlanta, Georgia, USA, 1999.