Network Decomposition for the Discrete Time–Cost Tradeoff Problem — Part 2: Network Decomposition and Computational Results

Can Akkan

College of Administrative Sciences and Economics, Koç University, Istinye, 80860, Istanbul, Turkey

cakkan@ku.edu.tr

Andreas Drexl, Alf Kimms

University of Kiel, Institut für BWL, Olshausenstr. 40, 24118 Kiel, Germany

drexl@bwl.uni-kiel.de, kimms@bwl.uni-kiel.de

Abstract

In this second part of research done on solving the discrete time-cost tradeoff problem, we present the methods used to decompose an AoN project network into subnetworks by removing some arcs. The objective is to minimize such arcs, while obtaining subnetworks that can be solved easily. Thus, the decomposition problem is modeled as a *graph partitioning problem* on a undirected graph $G(V, E)$ where each node $v \in V$ has a non-negative weight $w(v)$, which equals its number of modes minus 1. A feasible partition is one where total weight of the nodes of each subnetwork does not exceed an upper limit. Two types of partitioning heuristics (namely, *graph growing* and *graph reduction*) along with a local improvement heuristic are tested and experimental results are presented.

1 The Model

We are given an activity–on–node project network (directed acyclic graph) with activities V, and $E \subset V \times V$ is the set of precedence constraints. For each activity $j \in V$ we are given a set M_j of modes. The discrete time–cost tradeoff problem addresses finding a mode for each activitity such that a given deadline T can be met with minimum cost while respecting the precedence constraints. Since this problem is NP-Hard, a "divide–and–conquer" approach would be to decompose the project network into subnetworks that can be solved with relative ease. We will let H denote the index set for these subnetworks. Then, for each $h \in H$, $V^h \subseteq V$ denotes the set of nodes and $E^h \subseteq V^h \times V^h \subseteq E$ denotes the set of arcs in subnetwork h. Let $\bar{E}$ be the set of arcs in E which have start and end nodes in different subnetworks. For H to be a partition, we must have $\bigcup_{h \in H} V^h = V$, $V^h \cap V^{h'} = \emptyset$, for $h \neq h'$, and $\bar{E} \cup \bigcup_{h \in H} E^h = E$.

Since $\bar{E}$ corresponds to the relaxed constraints in the MIP formulation discussed in Part I, the objective of network decomposition is to minimize $|\bar{E}|$ subject to two types of constraints. Firstly, each subgraph h, $G(V^h, E^h)$ should be connected. Otherwise each component of the subgraph could be treated as a seperate subgraph without increasing

$\bar{E}$. Secondly, problems corresponding to the subnetworks should be easy to solve. To this end, we let the weight $w(v)$ of each node v be $|M_j| - 1$ and let $W^h = \sum_{v \in V^h} w(v)$, denote the total weight of subnetwork h. Since, the number of modes per activity is a strong indicator of solution difficulty, our second contraint is $W^h \leq b$ for all h.

Clearly, neither the objective nor the constraints of network decomposition depend on the direction of edges, hence decomposition can be done on the corresponding undirected graph. That is, we can model the decomposition problem described above as a *capacitated graph partitioning problem* on a undirected graph $G(V, E)$ where each node $v \in V$ has a non-negative weight $w(v)$.

2 Decomposition Heuristics

2.1 Graph Growing Heuristic

This heuristic creates one partition at a time. Let V_f denote the set of nodes that have not been included in any partition (free nodes). The heuristic starts a partition, h, by choosing a node from V_f. Then it includes one node at a time, say n, chosen from V_f such that n is adjacent to at least one node in V^h and $W^h + w(n) \leq b$ until no such node exists. Then it "closes" the current partition and starts a new one. In our implementations the choice of node n has been done using two different myopic rules. Let $E(v, A)$ denote the number of edges between a node n and a set of nodes A.

(1) MinMax rule first determines the set, C, of capacity feasible nodes that are adjacent to at least one node in current partition V^h. Then, the set $C' \subseteq C$ is determined which consists of nodes v such that $E(v, V_f)$ is minimum. Finally, the set $C'' \subseteq C'$ is determined that contains the nodes v such that $E(v, V^h)$ is maximum. If $|C''| > 1$ then, the choice is made randomly.

(2) MaxGain rule first determines the set C as in MinMax. Then, among the nodes in C, choses the one that gives the maximum gain, where gain associated with including a node n into a partition V^h is defined as $E(n, V^h) - E(n, V_f)$.

2.2 Graph Contraction Heuristic

This heuristic has two stages: "Coarsening" and "Uncoarsening". Coarsening iteratively merges nodes and works as follows: Let $G(V_0, E_0)$ denote the original graph where all edges have weight 0 and $G(V_i, E_i)$ denote the smaller graph obtained in iteration i. If vertices $V^v \subset V_i$ are merged to become node $v \in V_{(i+1)}$ then $w(v) = \sum_{j \in V^v} w(j)$. Similarly, the edges of v is the union of the edges of nodes in V^v. If nodes $V^e \subseteq V^v$ have an edge to the same node $k \in V_i \setminus V^v$, then the weight of edge $(v, k) \in E_{(i+1)}$ equals $\sum_{j \in V^e} w((j, k))$. Each iteration of graph coarsening is done by "maximal matching", that is finding a set of edges such that no two edges share the same end-node and the

set is maximal in the sense that no other edge can be added to the set without making two edges incident on the same node. During coarsening, in order for an edge to be collapsed, the total weight of its end-nodes must not exceed the upper bound b.

"Uncoarsening" is done by assigning the nodes in V_i^v that make up the multinode $v \in V_{(i+1)}$ in the coarse graph to the partition of v. This preserves both W^h for all h and $|\bar{E}|$.

The following are the rules used in choosing two adjacent nodes to be merged during coarsening: Largest Vertex Weight Rule, Largest Edge Weight Rule, Randomized Largest Vertex Weight Rule, Randomized Largest Edge Weight Rule.

2.3 Local Improvement Heuristic

The defacto benchmark algorithm for graph bisection problem has been the local search algorithm of Kernighan and Lin (KL)[2]. Hendrickson and Leland[1] extended the algorithm to improve a graph with an arbitrary number of partitions. The basic idea of KL is to allow non-improving 2-exchanges that lead to an improving k-exchange without searching the entire k-exchange neighbourhood. All KL-like algorithms use *gain* from moving a vertex v from its current partition to partition k, defined as: $gain^k(v) = E(v, V^k) - E(v, V^{p(v)})$, where, $p(v)$ is the partition of vertex v. Thus gain is the reduction in $|\bar{E}|$ due to moving a vertex to another partition. Our KL-like algorithm takes into account the constraint on partition–weight and allows eliminating (but not creating) partitions.

References

[1] B. Hendrickson, R. Leland (1993) A multilevel algorithm for partitioning graphs, Technical Report SAND93-1301, Sandia National Laboratory, Albuquerque, New Mexico.

[2] B. W. Kernighan, S. Lin (1970) An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49, 191-307.