# An Efficient A* Algorithm for the Directed Linear Arrangement Problem

DERCHIAN TSAIH
Nanhua University
Department of E-Commerce
32 Chung Keng Li, Dalin
Chiayi, TAIWAN
dtsaih@mail.nhu.edu.tw

GUANGMING WU
Nanhua University
Department of Information Management
32 Chung Keng Li, Dalin
Chiayi, TAIWAN
gmwu@mail.nhu.edu.tw

CHIEHYAO CHANG
Nanhua University
Department of E-Commerce
32 Chung Keng Li, Dalin
Chiayi, TAIWAN
cychangs@mail.nhu.edu.tw

SHAOSHIN HUNG
Wufeng Institute of Technology
Department of CS
117 Jianguo Rd, Minsyong
Chiayi, TAIWAN
hss@cs.ccu.edu.tw

CHINSHAN WU
Wufeng Institute of Technology
Department of E-Commerce
117 Jianguo Rd, Minsyong
Chiayi, TAIWAN
jackwu@mail.wfc.edu.tw

HUILING LIN
Ling Tung University
Department of International Business
1 Ling Tung Rd
Taichung, TAIWAN
ljoyce@mail.ltu.edu.tw

*Abstract:* In this paper we present an efficient A* algorithm to solve the Directed Linear Arrangement Problem. By using a branch and bound technique to embed a given directed acyclic graph into a layerwise $partition\ search\ graph$, the optimal directed ordering is then be identified through a A* shortest path search in the embedding graph. We developed a hybrid DC+BDS algorithm to approximate the optimal linear arrangement solution, which includes directed clustering and bidirectional sort technique. Along with a lower bound based on the maximum flow technique, this approximation solution is used as an upper bound to prune the state space during the A* search. In order to reduce the memory requirement of the A* search, we also discuss a implementation of the relay node technique from Zhou and Hansen [22].

*Key–Words:* Directed Linear Arrangement, Directed Clustering, A* Search

## 1 Introduction

Broadcasting data becomes a widely studied problem in the wireless environment due to its practical importance. Nowadays researchers are becoming increasingly interested in the study of correlated data broadcasting, which leading to a requirement to devise an efficient and effective scheduling algorithm for the correlated data. Applications are generally represented by directed acyclic graphs (DAGs), and solved via a Directed Linear Arrangement. Like the undirected correlated data broadcasting, for which many heuristic solutions have been proposed [20][4], this problem is hard in general, but has been solved in the case with a single root vertex, in which every other vertex either has a path directly to this root vertex, or can be reached from it [2].

Many heuristics had been proposed for the scheduling problem. By taking into account the system heterogeneity, Lai [12] proposed a duplication-based heuristic for scheduling tasks in a distributed heterogeneous environments. Liu [13] proposed a scheduling algorithm for an out-tree DAG based on task duplication. Adelson-velsky [1] proposed a polynomial time algorithm for scheduling tasks in AND-OR graphs. Wang [21] proposed a dynamic task scheduling algorithm in grid computing environment. Pan [15] adopted a heuristic method to schedule the resource constrained projects by incorporating fuzzy set theory to model the uncertain activity duration times.

Kaji presented a simulated annealing method for sequential partitioning of directed acyclic graphs [9], which is a variation of the scheduling problem discussed herein. Although the proposed algorithm is effective and efficient, it cannot guarantee to find an optimal solution, and is hard to fine-tune the cooling factors to specific problems. In a recent study for scheduling dependent tasks in a heterogeneous system, Sakellariou [17] presented a partition-based algorithm that determines the optimal vertex order by sorting all nodes with its rank value and additional local group optimization. However, the effectiveness in solving the directed ordering problem depends mainly on an optimal order of subset nodes within each local

group.

The structure of this paper is as follows. Section 2 presents the Directed Linear Arrangement Problem, and introduces a lower bound of cut before each placed vertex by the Max-Flow theorem. The third section presents a hybrid DC+BDS heuristic to approximate the DLA problem. Sections 4 presents a technique to solve the DLA problem using the A* search in an induced partition search graph. Section 5 presents the experimental results, and Section 6 concludes this study.

# 2 Directed Linear Arrangement Problem

Given a directed acyclic weighted graph $G$=($V$,$E$,$w$) with non-negative weights, the directed linear arrangement problem is to determine a surjection $f$: $V \longmapsto \{0,1,2,....|V|\text{-}1\}$ such that $(u,v) \in E \Longrightarrow f(u) < f(v)$, i.e. $f$ is a topological order which minimize the total weighted latency of edges. The total weighted latency of edges in the arrangement $f$ can be considered as follows:

$$W(f) = \sum_{(u,v)\in E} w_{u,v}(f(v) - f(u)) \qquad (1)$$

where $w_{u,v}$ denotes the weight of the edge from $u$ to $v$. Let $n = |V|$, $m = |E|$ and $W_i(f)$ denote the *edge cut* across between vertex in position $i$ and position $i + 1$ from $f$. Equation 1 can be rewritten as follows:

$$\begin{aligned} W(f) &= \sum_{i=1}^{n-1} W_i(f) \\ &= \sum_{i=1}^{n-1} \sum_{u,v\in V\&\&\, f(u)\leq\, i<f(v)} w_{u,v} \qquad (2) \end{aligned}$$

Let $Scut(A, B)$=$\sum_{u\in A,v\in B} w_{u,v}$ denote the total cut weight from vertex in $A$ to vertex in $B$. Since a valid topological order of vertex in a DAG has no backward edge, from Equation 2, for a valid topological order of vertex arrangement $f$, $W_i(f)$ is the cut weight from $\{j|f(j) \leq i\}$ to $\{j|i < f(j)\}$, which is given by

$$W_i(f) = Scut(\{j|f(j) \leq i\}, \{j|i < f(j)\})$$

Thus, under a topological order $f$, the cut weight across vertex in position $i$ and in position $i + 1$ is the cut weight from vertices arranged before position $i$, vertex in position $i$, to vertices arranged after position $i$.

**Example 1.** Consider a directed acyclic graph from Figure 1a, and one of its topological sort graph from Figure 1b, $f$(1)=3, $f$(2)=1, $f$(3)=4, $f$(4)=2 and $f$(5)=5. The cut weight between the first and second vertices is $W_1(f) = SCut$({2}, {4,1,3,5})=12; the cut weight between the second and third vertices is $W_2(f) = Scut$({2,4}, {1,3,5})=8, and so on. The total weighted latency of edges is thus 74.

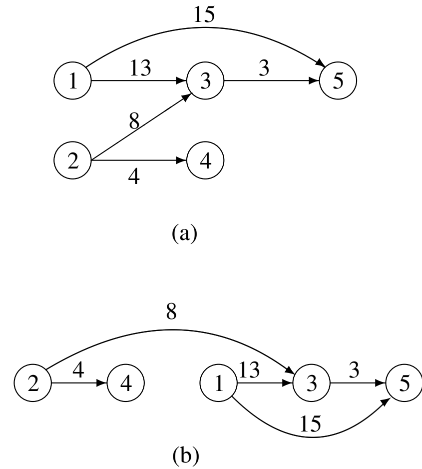


Figure 1. (a) A simple directed acyclic graph and (b) one of its corresponding TSG.

## 2.1 The Lower/Upper Vertex Set

Let $x \prec y$ denote the existence of a path from $x$ to $y$. The following definition is then introduced:

**Definition 1** *The partition of $V$ between two disjoint set of vertices, $V^l$ and $V^u$, is called a graph cut of acyclic directed graph $G(V, E)$. If no $x \prec y$ exists for $x \in V^u$, $y \in V^l$ and $V^u = V \setminus V^l$, then ($V^l$,$V^u$) is a directed vertex set pair and $V^l$ and $V^u$ are the lower and upper vertex sets of ($V^l$,$V^u$), respectively.*

For a directed pair ($V^l, V^u$) with nonempty $V^u$, if a vertex $x \in V^u$ exists such that ($V^l \bigcup\{x\}, V^u \setminus \{x\}$) is also a directed pair, then $x$ is called a *split vertex* of $V^u$. For detaching $x$ from $V^u$ to form another directed pair ($V^l \bigcup\{x\}$, $V^u \setminus \{x\}$), $x$ must be in the minimum source set of $V^u$.

**Definition 2** *Define $M(U)$ as the minimum source set of $U$ that itself is the subset of $U$, where either each vertex in $U$ is in this subset, or a path exists from vertices in this subset to rest of vertices in $U$.*

## 2.2 Max-Flow/Min-Cut Lower bound

Let $w_u^{(1)} \geq w_u^{(2)} \geq w_u^{(3)} \geq$ ... denote the weights of the outgoing edges from $u$. Thus, $X^+(u) = \sum_i i w_u^{(i)}$ gives the minimum total weight of latency of outgoing

edges from $u$. Similarly, let $X^-(u)$ denote the minimum total weight of latency of the incoming edges from $u$. Ganapathy[7] provide a lower bound for the directed linear arrangement problem as follows:

$$W(f) \geq \max\left(\sum_{u\in V} X^+(u), \sum_{u\in V} X^-(u)\right) \qquad \forall f$$

This technique is very efficient, since it only considers the order and weights of edges entering and leaving each vertex. However, instead of simply adding the edge weight for each vertex, the minimum cut weight before each vertex can be derived through the max-flow-min-cut theorem.

For each $u$ within a topological arrangement of vertex set, if the vertices before $u$ are represented as $V^l$, and $u$ and vertices after $u$ are represented as $V^u$, then the cut before $u$ in this topological arrangement is given by the set cut between $V^l$ and $V^u$. Let $Mcut(u)$ denote the minimum weight cut before $u$ in any topological arrangement. Obviously, the minimum cut before $u$ for any topological order is the minimum set cut between any directed $(V^l, V^u)$ pair with $u \in M(V^u)$, i.e.,

$$Mcut(u) = \min_{u\in M(V^u)} Scut(V^l, V^u) \qquad (3)$$

For $u \in M(V^u)$ it is seen that the vertices which have a path to $u$ must in subset $V^l$, and the vertices which are reachable from $u$ must in subset $V^u$. By using the same approach for hypergraph partition from Patkar [16], the solution of this minimum cut problem can be found through maximum flow problem with sources as the vertices which have at least one path to $u$, sinks as $u$ and vertices which are reachable from $u$. That is, under $u$ in $M(V^u)$, the minimum set cut between $(V^l, V^u)$ pair of $V$ is equal to the maximum flow between $\{v|v \prec u\}$ and $\{u\} \bigcup \{v|u \prec v\}$.

The problem of maximum flow between multiple sources and multiple sinks can be reduced to single source/single sink case by adding a super source which has one $\infty$ capacity outgoing edge to each source and adding a super sink which has one $\infty$ capacity incoming edge from each sink. Hence for each vertex $u$, $Mcut(u)$ can be found through the maximum flow between the super source $s$ and super sink $t$, where the super source $s$ has one $\infty$ capacity outgoing edge to each vertex which have a path to $u$ and the super sink $t$ has one $\infty$ capacity incoming edge from $u$ and from each vertex which $u$ has a path to it.

With minimum cut before vertex $u$ in any topological order as $Mcut(u)$, the summation of all minimum cut will give a better lower bound to directed linear arrangement problem as following.

$$W(f) \geq \sum_{u\in V} Mcut(u) \qquad \forall f$$

For each vertex in Example 1, after adding super source $s$ and super sink $t$, the minimum weight cut before the vertex can be discovered through the max-flow-min-cut theorem, which are $Mcut(3)$=36, $Mcut(4)$=4, $Mcut(5)$=18 and zero for all other. Figure 2 shows the flow network for vertex 3.

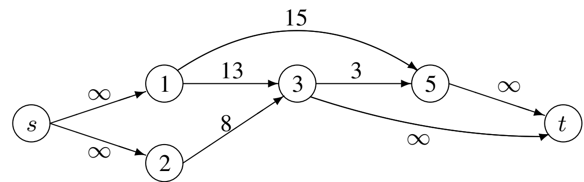


Figure 2. The flow network for minimum cut before vertex 3.

# 3 Composite Approximation Solution

## 3.1 Bidirectional Greedy Zero In-Degree Sort

The topological order of vertex can be computed efficiently either by depth-first sorting algorithm or zero in-degree sorting, which is based on breadth first traversal algorithm [3]. The greedy strategy can also used to find a topological order for the directed linear arrangement problem, by either depth-first or breadth-first traversal.

For a greedy strategy used in depth-first traversal algorithm, Sakellariou[17] proposed finding the vertex order by sorting vertices based on rank value. The rank value, $r_i$ , of a vertex $i$ is recursively defined as follows:

$$r_i = \max_{(i,j)\in E} (w_{i,j} + r_j) \qquad (4)$$

All the rank value can be found through Equation 4 with only a single depth-first traversal. Limited by its depth-first nature, the quality of this heuristic solution is greatly depend on the hierarchical structure of directed graph. Therefore, this work propose a breadth-first greedy algorithm based on the zero in-degree sort process.

Each step of the zero in-degree sort consists of two phases. The first phase is a vertex selection phase, in which a vertex $u$ with zero incoming edges is randomly selected and reported. The second phase is a vertex deletion phase, where vertex $u$ and all its outgoing edges are deleted from the graph. The optimum vertex order can then be built by selecting the optimum vertex at each vertex selection phase.

There are two heuristics for the selection of vertex during each step of the zero in-degree sorting

processes, which are maximum-incident heuristic and least-cost heuristic.

**• Maximum-Incident Heuristic**

Let $F_{in}(u) = \sum_{(v,u)\in E} w_{v,u}$ denote the total weight of edges which incident on $u$. By maximizing the total edge weight incident on the select vertex, the edge weight extended to the following vertices in the list is minimized. This leads to a simple greedy strategy, in which each zero in-degree sort selects the vertex with the largest $F_{in}(u)$.

**• Least-Cost Heuristic**

For a given partition pair ($V^l$, $V^u$), the total cut weight between $V^l$ and $V^u$ is $\sum_{v\in M(V^u)} F_{in}(v)$, and the minimum total cut weight for placing $V^u$ is as follows.

$$\begin{aligned} R(V^u) &= \sum_{v\in M(V^u)} F_{in}(v) + \min_u \sum_{v\in V^u, v\neq u} Mcut(v) \\ &= \sum_{v\in M(V^u)} F_{in}(v) + \sum_{v\in V^u} Mcut(v) \\ &\qquad - \max_u Mcut(u) \qquad (5) \end{aligned}$$

As shown in Equation 5, the selection with vertex with largest $Mcut(u)$ will minimize estimated remaining cost of finishing the ordering. This leads to the second greedy strategy, in which each zero in-degree sort selects the vertex with the largest $Mcut(u)$.

During the vertex selection phase of zero in-degree sort, the maximum-incident heuristic picks the vertex $u$ with largest $F_{in}(u)$, while the least-cost heuristic picks the vertex $u$ with largest $Mcut(u)$. From Equation 3, the minimum cut weight before vertex $u$ can be rewritten as follows.

$$Mcut(u) = F_{in}(u) + \min_{u\in M(V^u)} \sum_{x\in V^l, y\in V^u, y\neq u} w_{x,y} \qquad (6)$$

Therefore, although both heuristics use the similar greedy approach to select the zero in-degree vertex, the maximum-incident heuristic is a locally optimum approach which considers only the cut weight into the selected vertex, and the least-cost heuristic considers both the cut weight into and across the selected vertex. The effectiveness of both heuristics may vary with different structure of the directed graph. Thus, we can establish a greedy zero in-degree sort by applying either vertex selection heuristics, or both vertex selection heuristics and pick the better result from applying both heuristics.

While zero in-degree sort picks the vertex from the zero in-degree vertices and place each selected vertex into ordered list from the beginning of list, there is also a backward version of zero in-degree sort (considering the original zero in-degree sort as a forward version). The backward version of zero in-degree sort first reverses all its directed edges, then picks a vertex from the zero in-degree vertices, and places the selected vertex into the ordered list from end of list.

The maximum-incident and least-cost heuristic can also be applied in backward zero in-degree sorting, such that the backward zero in-degree sort with maximum-incident or least-cost heuristics forms the backward greedy zero in-degree sort.

Either by multiple zero in-degree vertices in a DAG (or multiple zero out-degree vertices in backward greedy zero in-degree sort) or sorting vertices with same $F_{in}(u)$( or $Mcut(u)$), the forward or backward greedy zero in-degree sort may randomly pick the vertex, leading to a less optimal vertex selection. However, the forward and backward sort can adopt each other as a tie breaking strategy, and the solution from the forward sort and the solution from the backward sort can be used as a preference list for each other.

The bidirectional sort algorithm iterating the back to back forward and backward greedy zero in-degree sort. Starting with random ordering as a preference list, the algorithm keep using results from other as a preference list, and the algorithm ends when the vertex order from both forward sort and backward sort are identical, or the solution is no longer improved. Thus, better heuristic than applying the forward or backward alone is provided. The next example is used to explain the details of bidirectional sort algorithm.

**Example 2.** Consider a directed acyclic graph from Figure 3, and an initial random reference list $f$ =[1, 2, 3, 4, 5, 6] with cost of 71. In the first round of algorithm, after the forward sort, both maximum-incident heuristic and least-cost heuristic report the same order of $f$ =[1, 2, 3, 4, 6, 5] with cost of 66. Since the vertex order is changed and solution is improved, $f$ is reversed and used as the preference list in the backward sort. After the backward sort both greedy heuristics report the same order of $f$ =[5, 4, 1, 6, 3, 2] with cost of 33. The $f$ is then reversed into [2, 3, 6, 1, 4, 5], in which the vertex order is changed and solution improved. Thus, the bidirectional sort algorithm enters the second round with new $f$ as the preference list. In this case, both greedy heuristics in the forward sort report the same order of $f$ =[2, 3, 6, 1, 4, 5], in which the vertex order is the same as the preference list. Therefore, the algorithm ends, and reports $f$ =[2, 3, 6, 1, 4, 5].

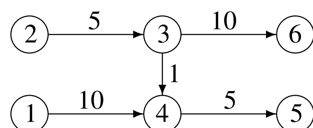


Figure 3. Directed acyclic graph of Example 2.

### 3.2 Hierarchical Agglomerative Clustering with Directed Merge

Agglomerative hierarchical clustering is a bottom-up clustering method where clusters have sub-clusters, which in turn have sub-clusters, etc. The algorithm computes a complete hierarchy of clusters and does not require the number of clusters to be known in advance. Start with every single vertex in a single cluster. Then, in each successive iteration, it merges the closest pair of clusters by satisfying some similarity criteria, until all of the vertices are in one cluster [11].

Hierarchical agglomerative clustering with directed merge is the directed version of HAC, or directed HAC. It concerns grouping vertices in pairs based on their similarity and/or their proximity. Directed merge links two sequential vertex sets in the source and end clusters to form a new sequential set of vertices in the source cluster before the vertices in the end cluster in the ordering.

As with the unordered HAC, each cluster can be a source or end cluster for directed merging. However, for each source/end cluster pair, there shall not exist any edges starting from end cluster to source cluster, and shall not exist any path starting from vertex in source cluster through other cluster and destined to the vertex in the end cluster.

To determine whether one source/end cluster pair can be directed merged without compromising the acyclicity, these two clusters are first test-merged, and the topological sort (either with depth first search or zero in-degree sort) is used to validate the acyclicity after the test merge. If the acyclicity is not violated after this test merge, then these source/end cluster pair is considered as directed-mergeable, and not otherwise.

In each iteration, the pair of clusters with the highest cohesion is merged by average-link clustering [8][6]. The similarity between each pair of clusters is measured according to the group average directed weight between them, which is an objective function based on the sizes of the clusters being merged, along with the directed edge weight from source cluster to end cluster [10]. The group average directed weight from cluster $\Gamma$ to $\Delta$ is defined as follows.

$$W_c(\Gamma, \Delta) = \frac{1}{|\Gamma||\Delta|} \sum_{u \in \Gamma, v \in \Delta} w_{u,v}$$

Starting with each vertex as a singleton cluster, the directed-mergeability and group directed weight of each source/end cluster pair are pre-validated and computed. All cluster pairs are then sorted with a heap. The cluster pair which are not directed-mergeable is always on the bottom of heap, and the directed-mergeable cluster pair with the largest group directed weight is always on the top of heap. At each merging process, the directed-mergeable cluster pairs with largest group directed weight between them, is linked and directed merged into a new cluster with a linked vertex list from the joining of linked vertex lists of source and end clusters[5].

After the cluster pair is merged, the end cluster is removed, and the source cluster is rewritten by the new cluster created by merging. Then, each cluster $x$/removed cluster pair will merged into cluster $x$/new cluster pair, and each removed cluster/cluster $x$ pair will merged into new cluster/cluster $x$ pair. With all source/end cluster pairs pre-sorted into a heap, each merging process consists of the following steps:

1. Directed merge the source/end cluster pair from top of heap.
2. Remove the merged source/end cluster pair from the heap.
3. Update the heap.

After $n - 1$ directed merges, all clusters are then merged into one single cluster, and the directed clustering solution is determined from the linked vertex list of the last remaining cluster.

### 3.3 Hybrid Algorithm of Directed HAC and Bidirectional Sort

The clustering algorithm suffers from their inability to perform adjustments once the merging decision is made. However, several studies have reported that the clustering process can increase the average degree of vertices (or graph density), and improve the approximation solution from iterative partition heuristic [18] and simulated annealing placement [19].

The hybrid algorithm of directed clustering and bidirectional sort, called the DC+BDS, iteratively applied the directed clustering to the input graph, while between each pair of clustering processes the processing graph was duplicated and the bidirectional sort was applied on the duplicated graph to explore each vertex order solution.

Using this exhaustive search the DC+BDS algorithm returned the best vertex order after the end of last clustering process. By fully combining all benefits of directed clustering and bidirectional sort, the total weighted latency of edges can be minimized.

# 4 The Partition Search Graph

## 4.1 Mapping from a Directed Acyclic Graph

A $partition\ search\ graph$ $\widetilde{G}(\widetilde{V},\widetilde{E},\widetilde{w})$ is a directed acyclic graph induced from another directed acyclic graph $G(V,E,w)$, where $\widetilde{V}$ represents a node set(comparable to vertex set $V$); $\widetilde{E}$ represents an arc set(comparable to edge set $E$), and $\widetilde{w}$ represents the node weight, such that each node from $\widetilde{V}$ is mapped to a lower/upper partition pair of $V$; each arc from $\widetilde{E}$ is mapped to one state transition between two ordered partition pairs, and the node weight is mapped to the set cut between this ordered partition.

Let $\widetilde{n}$=$|\widetilde{V}|$, $\widetilde{m}$=$|\widetilde{E}|$, and $F_{out}(u) = \sum_{(u,v)\in E} w_{u,v}$ denote the total weights of edges which incident out from $u$. A $partition\ search\ graph$ has the following properties:

1. The graph has only a source node, denoted as $root$, that mapped to partition pair ($\emptyset$,$V$) and a sink node, denoted as $goal$, that mapped to partition pair ($V$,$\emptyset$).
2. Each node in $\widetilde{V}$ mapped to a lower/upper partition pair of $V$ and each arc in $\widetilde{E}$ mapped to a $split\ vertex$ from node in $\widetilde{V}$. Hence, $\widetilde{n}$=$\sum_{(V^l,V^u)} 1$ and $\widetilde{m}$ = $\sum_{(V^l,V^u)} |M(V^u)|$.
3. The node which mapped to a partition pair $(V^l, V^u)$ can be classified into a layer$-|V^u|$ group, such that all arcs that start from node in the layer$-k$ group must end in a node in the layer$-k$+1 group.
4. The node weight for a node which mapped to the partition pair ($V^l$, $V^u$) is $Scut$($V^l$, $V^u$). The node weight for a node mapped to ($V^l \bigcup\{a\}$, $V^u \backslash \{a\}$) can be determined through $Scut$($V^l$, $V^u$) as follows.

   $$Scut(V^l \bigcup\{a\}, V^u\backslash\{a\}) = Scut(V^l, V^u) - F_{in}(a) + F_{out}(a).$$

5. The minimum cost of a path, starting from a node which mapped to partition pair ($V^l$, $V^u$) to $goal$, is $\sum_{u\in V^u} Mcut(u)$.

The optimal order of $V$ then depends on every nodes mapped to ($V^l$, $V^u$) ordering its lower vertex set $V^l$ perfectly. The perfect ordering vertices of $V^l$ in the layer$-k$+1 group nodes requires perfectly ordered vertices of $V^l$ in the layer$-k$ group node. Starting from the layer$-0$ $root$ up to the layer$-n$ $goal$, the optimum linear order of vertices can be found through the shortest path between $root$ and $goal$. The DLA problem can thus be solved by a shortest path problem in this layered graph, with the cost function given by the sum of set cut along each path.

If all node layers fit in memory, then the $partition\ search\ graph$ can be constructed through the depth first traversal algorithm with a complexity of O($\widetilde{n}+\widetilde{m}$). Figure 4 shows the induced $partition\ search\ graph$ from Example 1. The $M(V^u)$ is shown inside each node; the $split\ vertex$ of each node is shown beside each arc, and the node weight for each partition cut is shown beside each node.

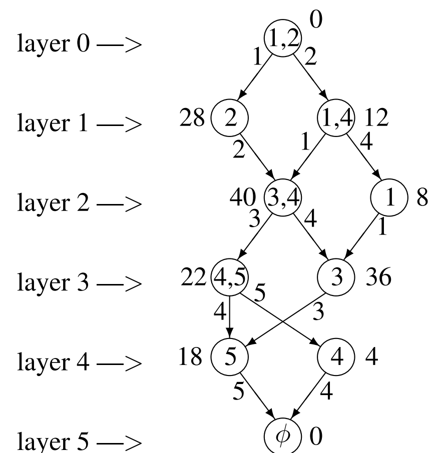


Figure 4. $Partition\ search\ graph$ from Example 1.

Let $h_{path}$ denote the number of vertex pairs $(x, y)$ with $x \prec y$. For each $(V^l, V^u)$ pair there exists one corresponding $M(V^u)$, and if $x \prec y$, then $x$ and $y$ do not coexist in the same $M(V^u)$. Therefore, the number of single element $M(V^u)$ is $n$, and for $i \geq 2$, the number of $i$ elements $M(V^u)$, under the general independent assumption, is $\binom{n}{i}(1-\frac{\binom{n-2}{i-2}}{\binom{n}{i}})^{h_{path}}$. The average number of valid $(V^l, V^u)$ pair, denoted by $\widetilde{n_{avg}}$, is given by

$$\begin{aligned}\widetilde{n_{avg}} &= n+\sum_{i=2}^{n}\binom{n}{i}\left(1-\frac{\binom{n-2}{i-2}}{\binom{n}{i}}\right)^{h_{path}} \\ &= n+\sum_{i=2}^{n}\binom{n}{i}\left(1-\frac{i(i-1)}{n(n-1)}\right)^{h_{path}}\end{aligned}$$

### 4.2 A* Search with Divide-and-Conquer Solution Reconstruction

A* is a best-first search algorithm that finds the least-cost path from a given initial node to one goal node. The basic idea of A* algorithm is to systematically go through a list of nodes which is ordered according to their estimated total cost to goal node.

There are four node variables in each node $x$ which mapped to the $(V^l, V^u)$ partition pair: $pred(x)$ denotes the backtrack pointer to the upper layer of node $x$, $s(x)$ denotes the node weight of $x$, $g(x)$ denotes the minimum cost of path from the $root$ to $x$(not include $s(x)$) and $h(x)$ denotes the minimum cost of path from $x$ to $goal$. For a node $x$ which mapped to the $(V^l, V^u)$ partition pair, $pred(x)$ is used for backtracking shortest path to $root$, and $s(x)$, $g(x)$, $h(x)$ are used to record values of follows.

$$s(x) = Scut(V^l, V^u)$$

$$g(x) = \sum_{y \neq x, y \in sp(x)} s(y)$$

$$h(x) = \sum_{a \in V^u} Mcut(a)$$

where $sp(x)$ denotes the shortest path from $root$ to $x$. Starting from the layer$-0$ $root$ with $s(root)$= $g(root)$ =0, $h(root) = \sum_{u \in V} Mcut(u)$, for a layer$-k$ node $x$ which mapped to $(V^l, V^u)$ pair, a layer$-k$+1 node $y$ which mapped to $(V^l \bigcup \{a\}, V^u \setminus \{a\})$ pair, node $x$ updates node $y$ with follows.

1. $s(y) = s(x) - F_{in}(a) + F_{out}(a)$
2. $h(y) = h(x) - Mcut(a)$
3. If $s(x) + g(x) < g(y)$ then $g(y)$ is updated with $g(x) = s(x) + g(x)$, and $pred(y)$ is updated with $pred(y) = x$

The admissible function $h(x)$ is used as the heuristic estimate of cost starting from $x$ to reach $goal$. Since $h(x)$ is the lower bound of the actual cost to reach $goal$, the algorithm can guarantee finding the optimal solution. During the shortest path search each node $x$ with $g(x) + h(x)$ larger than the upper bound of the DC+BDS approximation solution is pruned to reduce the search space.

The cumulated cost and the backtrack pointer of each node are updated layer by layer, and this search method can be considered as a breadth-first search algorithm. By using the breadth-first search, the optimal path to each node has been found before the expansion to next layer. The algorithm traverses various paths from $root$ to $goal$. By keeping all the layers of the search graph in memory, the shortest path from $root$ to $goal$ can be recovered starting from $pred(goal)$.

Each branch with less optimal solution than the DC+BDS upper bound can be pruned from the search graph. Therefore, the efficiency of the A* algorithm is determined by the difference between the upper bound from the DC+BDS approximation solution, and the lower bound from the max-flow-min-cut technique. Figure 5 shows the search results for using A* algorithm on Example 1. For each node $x$ mapped to partition pair $(V^l, V^u)$, $M(V^u)$ is shown inside the node; the $(s(x), g(x), h(x))$ triplex is shown beside the node, and each $split\ vertex$ is shown beside the backtrack pointer. The search result shows that the optimal path is [2, 4, 1, 3, 5 ] with cost 74.

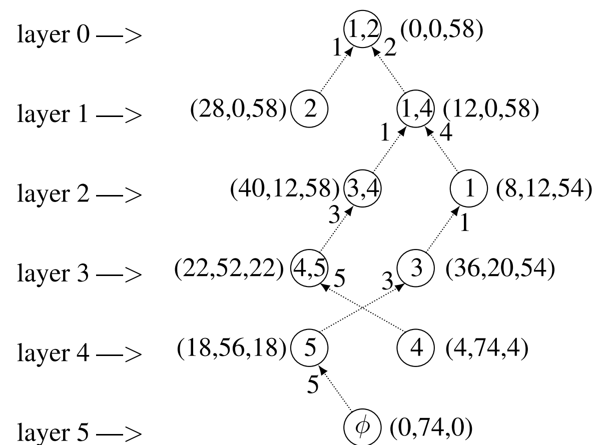


Figure 5. The search result from using A* algorithm on Example 1.

However, since the graph may grow exponentially, this approach cannot be used in a sparse memory system, and can only be used in a small DAG or in a subset of complete DAG returned from the partition-approach algorithm. In a sparse memory system each layer of nodes can not be stores in memory at same time. Zhou and Hansen [22] described a memory-efficient approach to graph searching called $Breadth\text{-}First\ Heuristic\ Search$(BFHS), which can be used in a partition search graph to reduce the memory requirement.

Starting from layer$-0$ $root$, the layer$-k$ nodes are used to update the cost and backtrack pointers for layer$-k$+1 nodes. Unlike the conventional backtrack pointer in the breadth-first search that points to the best layer$-k$+1 ancestor node, each nodes in layer$-k$ points to layer$-0$ $root$ for $k < \frac{n}{2}$, and to its best layer$-\frac{n}{2}$ ancestor node for $k > \frac{n}{2}$.

Since all backtrack pointers are either point to $root$ or a layer$-\frac{n}{2}$ node, by retaining all layer$-\frac{n}{2}$ nodes in memory, each layer$-k$ nodes can be free from memory after all updates of layer$-k$+1 nodes are completed.

After the layer$-n$ $goal$ is constructed and completely updated, a layer$-\frac{n}{2}$ $relay\ node$, which is within the optimal path to $goal$, can be found from $pred(goal)$. The $relay\ node$ is used to divide the search problem into two sub-problems, namely finding the optimal path from $root$ to the relay node, and finding the optimal path from the relay node to $goal$. Both sub-problems are then solved through the original search algorithm in order to find the new $relay\ nodes$ between $root$ and the old $relay\ node$, and between the old $relay\ node$ and $goal$. This process continues recursively until all nodes in the optimal path are identified.

By combining the divide-and-conquer strategy and A* algorithm with pruning capability, nodes can be constructed and updated layer by layer without saving the complete search graph in memory.

# 5 Experimental Results

In this section all proposed algorithms were experimentally compared by measuring their performance over a large number of randomly generated DAGs with multiple-source and multiple-sink. To evaluate the performance of the algorithms over a wide range of data characteristics, the edge weight of each generated DAG followed a generalized Zipf distribution. Let $w^{(i)}$ denote weight of an edge ranked in the $i$th position of sorted edge list. The generalized Zipf distribution is defined as follows.

$$w^{(i)} \propto \frac{1}{i^s} \qquad 1 \leq i \leq m$$

where $s$ is the $skew$ coefficient. For $s$=0, the Zipf reduces to uniform distribution with $w^{(i)} = \frac{1}{m}$, whereas larger values of $s$ derive increasingly skewered distributions. Each data set of simulation experiments is from 20 randomly generated DAGs, and is evaluated for $n$=50, for $m$ varying from 100 to 300, and for skew factor $s$ in the range 0-2.0.

Table 1 compares the performances of 6 methods; those results are also compared with the optimum arrangement solutions from the shortest path search from partition search graph. The first method is the lower bound from $h(root)$. Method 2 and 3 involve the forward sort algorithm with maximum-incident heuristic and least-cost heuristic, respectively. Method 4 and method 5 are bidirectional sort and directed clustering, respectively. Method 6 is the hybrid DC+BDS algorithm.

**Lower bound.** The lower bound which utilizes the max-flow-min-cut theorem, became closer to the optimal solution as the number of edges increased, or the edges had an increasingly skewered distribution.

**Comparison between maximum-incident and least-cost heuristic.** The least-cost heuristic outperformed the maximum-incident heuristic for an edge weight with a skewered distribution, but performed worse with a more uniform edge weight distribution.

**DC+BDS algorithm.** Bidirectional sort algorithm implements the iterative improvement technique with both maximum-incident and least-cost heuristic, and outperforms the algorithm using either only the maximum-incident heuristic or least-cost heuristic. Both bidirectional sort and directed clustering were found to be vulnerable when the number of edges was small, or the edges had a skewered distribution. However, the simulation results show that the hybrid DC+BDS algorithm improves not only the arrangement solution but also the vulnerability of input graph skewness.

# 6 Conclusions

This work present a branch and bound method to convert the directed linear arrangement problem to a shortest path search problem, and solved with a A* algorithm with an upper limit calculated from a hybrid DC+BDS algorithm and a lower limit obtained from the max-flow-min-cut theorem. If the available memory is insufficient to store an entire layer of nodes, the solution can also be found using a partial search graph with only the promising nodes in each layer.

Additionally, analytical results show that the DC+BDS algorithm, which uses the directed clustering, in conjunction with the bidirectional sort method, achieves good quality approximate solutions for a wide range of edge numbers and skew ratios of edge weights.

*References:*

[1] G.M. Adelson-velsky, A. Gelbukh, E. Levner, A Fast Scheduling Algorithm in AND-OR Graphs, *Applied and Theoretical Mathematics and Computer Science, WSEAS Press*, 2001, pp. 170-175.

[2] D. Adolphson and T.C. Hu, Optimal linear ordering, *SIAM Journal on Applied Mathematics*, Vol 25, No 3, 1973, pp. 403-423.

[3] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *Data Structures and Algorithms*.

[4] Y.D. Chung and M.H. Kim, On Scheduling Wireless Broadcast Data, *Technical Report CS-TR-98-134*, KAIST, Department of Computer Science, 1998.

Table 1: Comparison of Simulation Results from different algorithms

| | $\lvert E\rvert$ =100 | | $\lvert E\rvert$ =150 | | $\lvert E\rvert$ =200 | | $\lvert E\rvert$ =250 | | $\lvert E\rvert$ =300 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | opt=8.77 | | opt=11.53 | | opt=12.65 | | opt=13.82 | | opt=14.40 | |
| Max-Flow-Min-Cut Lower Bound | 5.56 | (-36.49%) | 9.34 | (-19.00%) | 11.20 | (-11.41%) | 12.92 | (-6.48%) | 13.78 | (-4.32%) |
| Maximum-Incident | 10.40 | (+18.67%) | 12.16 | (+5.49%) | 12.93 | (+2.26%) | 13.94 | (+0.92%) | 14.48 | (+0.51%) |
| Least-Cost | 10.98 | (+25.25%) | 12.38 | (+7.38%) | 13.19 | (+4.25%) | 14.02 | (+1.51%) | 14.56 | (+1.06%) |
| Bidirectional Sort | 9.37 | (+6.91%) | 11.75 | (+1.90%) | 12.80 | (+1.20%) | 13.91 | (+0.70%) | 14.43 | (+0.20%) |
| Directed Clustering | 9.80 | (+11.86%) | 12.21 | (+5.89%) | 13.13 | (+3.82%) | 14.02 | (+1.47%) | 14.55 | (+1.04%) |
| DC+BDS | 9.17 | (+4.60%) | 11.69 | (+1.42%) | 12.73 | (+0.62%) | 13.86 | (+0.34%) | 14.43 | (+0.18%) |

(a) $s$=0 , $\lvert V\rvert$=50

| | $\lvert E\rvert$ =100 | | $\lvert E\rvert$ =150 | | $\lvert E\rvert$ =200 | | $\lvert E\rvert$ =250 | | $\lvert E\rvert$ =300 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | opt=6.15 | | opt=8.73 | | opt=9.72 | | opt=10.33 | | opt=10.73 | |
| Max-Flow-Min-Cut Lower Bound | 4.49 | (-26.97%) | 7.42 | (-14.99%) | 8.84 | (-9.06%) | 9.82 | (-4.98%) | 10.38 | (-3.17%) |
| Maximum-Incident | 8.62 | (+40.30%) | 10.23 | (+17.15%) | 10.44 | (+7.40%) | 10.86 | (+5.04%) | 11.06 | (+3.09%) |
| Least-Cost | 8.65 | (+40.71%) | 10.25 | (+17.32%) | 10.55 | (+8.50%) | 10.66 | (+3.15%) | 10.90 | (+1.59%) |
| Bidirectional Sort | 7.08 | (+15.27%) | 9.20 | (+5.34%) | 10.00 | (+2.84%) | 10.47 | (+1.29%) | 10.82 | (+0.89%) |
| Directed Clustering | 7.90 | (+28.55%) | 9.50 | (+8.82%) | 10.41 | (+7.03%) | 10.72 | (+3.75%) | 10.97 | (+2.26%) |
| DC+BDS | 6.45 | (+4.93%) | 8.98 | (+2.88%) | 9.89 | (+1.69%) | 10.41 | (+0.75%) | 10.79 | (+0.57%) |

(b) $s$=1.0 , $\lvert V\rvert$=50

| | $\lvert E\rvert$ =100 | | $\lvert E\rvert$ =150 | | $\lvert E\rvert$ =200 | | $\lvert E\rvert$ =250 | | $\lvert E\rvert$ =300 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | opt=5.18 | | opt=9.58 | | opt=10.90 | | opt=11.34 | | opt=11.84 | |
| Max-Flow-Min-Cut Lower Bound | 4.75 | (-8.15%) | 9.08 | (-5.26%) | 10.60 | (-2.75%) | 11.13 | (-1.89%) | 11.73 | (-0.92%) |
| Maximum-Incident | 9.36 | (+80.76%) | 12.38 | (+29.10%) | 12.27 | (+12.48%) | 12.73 | (+12.27%) | 12.57 | (+6.21%) |
| Least-Cost | 9.38 | (+81.22%) | 11.64 | (+21.42%) | 11.44 | (+4.90%) | 11.88 | (+4.79%) | 12.08 | (+2.00%) |
| Bidirectional Sort | 6.03 | (+16.57%) | 10.17 | (+6.08%) | 11.40 | (+4.56%) | 11.49 | (+1.35%) | 11.93 | (+0.78%) |
| Directed Clustering | 11.14 | (+115.19%) | 11.71 | (+22.12%) | 11.78 | (+8.01%) | 12.29 | (+8.34%) | 12.37 | (+4.53%) |
| DC+BDS | 5.41 | (+4.50%) | 9.99 | (+4.24%) | 11.15 | (+2.27%) | 11.48 | (+1.24%) | 11.92 | (+0.65%) |

(c) $s$=2.0 , $\lvert V\rvert$=50


[5] Y.D. Chung, S. Bang, M. Kim, An efficient broadcast data clustering method for multipoint queries in wireless information systems, *The journal of Systems and Software*, 2002, pp. 173-181.

[6] C. Ding, X. He, H. Zha, M. Gu, H. Simon, A Min-max cut algorithm for graph partitioning and data clustering. *IEEE 1st Conference on Data Mining*, 2001, pp. 107-114.

[7] M. Ganapathy, S. Lodha, On Minimum Circular Arrangement, *Lecture Notes in Computer Science*, Vol 2996, 2004, pp. 394-405.

[8] A.K. Jain, M.N. Murty, P.J. Flymn, Data Cluster: a review. *ACM Computing Surveys*, Vol 31, No 3, 1999, pp. 264-323.

[9] T. Kaji, A. Ohuchi, A simulated annealing algorithm with the random compound move for the sequential partitioning problem of directed acyclic graphs, *European Journal of Operational Research*, Vol 112, No 1, 1999, pp. 147-157.

[10] D. Karger, Global Min-cuts in RNC and other ramifications of a simple Min-cut Algorithm, *Proc. ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 21-30.

[11] S. Kotsiantis, P. Pintelas, Recent Advances in Clustering: A Brief Survey, *WSEAS Transactions on Information Science and Applications*, Vol 1, No 1, 2004, pp. 73-81.

[12] K.C. Lai, C.M. Lee, J.F. Fang, A Critical Predecessor Duplication Scheduling Algorithm for Distributed Heterogeneous Computing Environments, *Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics*, 2006, pp. 497-502.

[13] Z. Liu, B. Fang, Y. Zhang, J. Tang, A Scheduling Algorithm for an Out-Tree DAG, *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, Vol 1, 2000, pp. 327-328.

[14] E. Milkova, Combinatorial Optimization: Mutual Relations among Graph Algorithms, *WSEAS Transactions on Mathematics*, Issue 1, Vol 7, January 2008, pp. 293-302.

[15] H. Pan, R.J. Willis, C.H. Yeh, Resource-constrained Project Scheduling with Fuzziness, *In Advances in FUZZY Systems and Evolutionary Computation, WSEAS Press*, 2001, pp. 173-179.

[16] S. Patkar, H. Sharma, H. Narayanan, Efficient Network Flow based Ratio-cut Netlist Hypergraph Partitioning, *WSEAS Transactions on Circuits and Systems*, Vol. 3, No. 1, January 2004, pp. 47-53.

[17] R. Sakellariou, H. Zhao, A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. *Proceedings of 13th Heterogeneous Computing Workshop*, Santa Fe, NM, 2004, pp. 111-124.

[18] Y.G. Saab, A Contraction-based ratio-cut Partitioning Algorithm, *VLSI Design*, Vol 15, No 2, January 2002, pp. 485-489.

[19] W.J. Sun, C. Sechen, Efficient and effective placement for very large circuits, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Issue 3, Vol 14, Mar 1995, pp. 349-359.

[20] D. Tsaih, G.M. Wu, C.B. Wang, Y.T. Ho, An Efficient Broadcast Scheme for Wireless Data Schedule Under a New Data Affinity Model, *Lecture Notes in Computer Science*, Vol 3391, 2005, pp. 390-400.

[21] D.Z. Wang, J.S. Zhan, F. Wan, L. Zhu, A Dynamic Task Scheduling Algorithm in Grid Environment. *Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics*, 2006, pp. 273-275.

[22] R. Zhou, E. Hansen, Breadth-first heuristic search, *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, Whistler, British Columbia, 2004, pp. 92-100.