Design and Analysis of Physical Design Algorithms*

Majid Sarrafzadeh   Elaheh Bozorgzadeh   Ryan Kastner   Ankur Srivastava

Computer Science Department
University of California, Los Angeles
Los Angeles, California 90095-1596
(Contact: majid@cs.ucla.edu)

ABSTRACT

We will review a few key algorithmic and analysis concepts with application to physical design problems. We argue that design and detailed analysis of algorithms is of fundamental importance in developing better physical design tools and to cope with the complexity of present-day designs.

1. INTRODUCTION

Problems in physical design are getting more complex and are of fundamental importance in solving present-day design problems. Full understanding of the problems and algorithm analysis of them are essential to making progress. Whereas problems are getting harder (e.g. by need for concurrent optimization, finer geometries, and larger problems) algorithms to cope with them have not been designed at the same rate.

A number of fundamental physical design algorithms have been developed in the past three decades. Examples are maze running and KL-FM partitioning [7, 8, 23, 24]. They are both in the heart of current CAD tools. A number of existing techniques, used heavily in current CAD tools, have not been fully analyzed. For example quadratic programming and hierarchical placement are used purely as a heuristic and their analysis is still open. Yet there are problems that are far from being understood. Researchers have started only very recently to study them. Examples are congestion estimation and minimization during placement.

A heuristics is a good starting point for solving a problem, however, it normally fails to perform well in a changing or a more complex scenario (e.g., hot-spot removal by annealing). To continue making effective CAD tools, we need to study problems deeper, analyze them thoroughly, and base the proposed heuristics on the performed analysis.

Here we will look at several algorithm design and analysis tools and concepts. The methods and concepts that we will point out in this paper are used less frequently in physical design tools.

This paper is organized as follows. In Section 2, it is shown how problem transformation is used to devise a new algorithm. Section 3 explains how proof of NP-Completeness of hard problems is useful in generating efficient algorithms to solve those problems. In Section 4, we explain how to obtain the power of a greedy method through the proof of its correctness. In Section 5, we describe that more global view to a problem can help improve the greedy algorithms. Approximation algorithms and advantages of analyzing the performance of heuristic methods are explained in Section 6. In Section 7, probabilistic algorithms and their ability to provide solution quality bounds are presented. In Section 8, some conclusions are given.

2. ON PROBLEM TRANSFORMATION: UPPER-BOUND ANALYSIS

Problem transformation is an effective methodology for solving a problem. Mathematicians have used transformation for many years and more recently by algorithm designers. Problem transformation can be used to devise a new algorithm (upper-bound) or to prove the complexity of a problem (lower-bound). In this section we will give an example of n problem transformation.

The graph-partitioning problem is to partition the vertices of a graph in k into roughly equal parts, such that the number of edges connecting vertices in different parts is minimized. In this paper, to simply the presentation, we use a graph model.

Formally, a graph $G = (V, E)$ is defined as a set of vertices V and a set of edges E, where each edge is a subset of the vertex set V. The graph partition problem is NP-complete. Recently, a number of researchers have investigated a class of algorithms that can give a reasonably good solution for the bi-partition problem [7, 8, 12, 13]. Most of these algorithms are more or less based on the FM algorithm, which was first proposed by Fiduccia and Mattheyses in 1982 [7]. FM algorithm is a very effective heuristic for the bi-partition problem. However, algorithm designers are more and more interested in the general k-way partition problem where k is greater than two.

When *k* is greater than 2, there are three typical methods to solve the *k*-way partition problem. Method A is a direct extension of 2-way FM-like algorithms. In the 2-way FM-like algorithms, each node can be moved to only one definite destination partition. A gain will be associated to this move. In the *k*-way partition problem, each node has *k-1* possible destination partitions. Thus method A is based on the 2-way FM algorithm while allowing moving any node to any of the *k-1* partitions. A node is picked to move if it results in the biggest gain in the total cut-cost. Method B is a all-way bi-partition improvement. It starts with an initial (and arbitrary) *k*-way partition. It picks two partitions from the total *k* partitions at a time and performs bi-partitioning to improve the all-way cut-cost between these two partitions. Finally, method C is a hierarchical approach. It will recursively bi-partition the target graph until we have *k* partitions.

Suaris and Kedem used method A on a 4-way partition problem [14, 15]. For *k* greater than 4, this method A is rarely used since it needs a lot of memory and is slow. In practice, the all-way bi-partition approach (Method B) and the hierarchical approach (Method C) are often used. The question is that which one is a better way to solve the *k*-way partition problem. In the original Kernighan-Lin's paper [8], the author argues that the hierarchical approach is a very greedy method. Intuitively, the hierarchical approach cannot recover the possible mistakes made by the previous cuts. Thus it will easily get stuck into a local minimum. They also argue that given enough time, the all-way bi-partition method should be able to explore most part of the solution space. Thus it has a better chance to get a good result.

Theoretical analysis is done on the all-way bi-partition approach and the hierarchical approach assuming we use a $?$-approximation bi-partition heuristic. If the optimal cut cost for a *k*-way partition problem is *C*, we prove that the cut cost from the hierarchical approach has an upper bound of $?C?\log k$ while the cut cost from the all-way bi-partition approach has an upper bound of $?C?k$ where *k* is the number of partitions.

Thus, not only a multi-way partitioning problem can be transformed to a sequence of 2-way (and thus, make use of available algorithms and tools for 2-way partitioning), it can be proved that the final solution is provably good.

3. PRACTICAL IMPLICATION OF LOWER-BOUND ANALYSIS

Stephen Cook first introduced the theory of NP-Completeness in a paper, 1971, entitled "The Complexity of Theorem Proving Procedures" [20]. He proved that one particular problem in NP, called the *satisfiability problem* has the property that every problem in NP can be can be polynomially reduced to it. Hence if the *satisfiability problem* can be solved polynomially, so can every other problem in NP. Also if one problem in NP is intractable, then *satisfiability problem* must also be intractable. Subsequently Richard Karp in 1972 proved that the decision problems of various combinatorial optimization problems (like traveling salesman) are just as hard as *satisfiability problem (SAT)* [21]. Since then a lot of problems have been proved to be equivalent in difficulty to these problems. Such problems, which

are the hardest in the set NP are called NP-Complete problems. The big question which is still open in the field of mathematics and computer science is whether or not NP-Complete problems are intractable. Although no proof exists about the implications of NP-Completeness on intractability, the knowledge of problem being NP-Complete suggests that a major breakthrough will be needed to solve it with a polynomial time algorithm. For more details on the theory of NP-Completeness we refer the reader to [18].

Many problems in VLSI-CAD are NP-Complete. Since NP-Completeness suggests that finding a polynomial time algorithm for the problem is unlikely, most people in our community have begun to somewhat underestimate the potential of proving problems NP-Complete. We strongly believe that knowledge of NP-Completeness can have two significant impacts on the course of research. First it points out that the research must be directed towards finding efficient heuristics and not optimal polynomial time algorithms. Second, the whole process of proving problems NP-Complete gives a deep insight into the problem. The actual reason because of which the problem gets NP-Complete becomes known. This information along with the discovery of relationship between problems (the NP-Complete problem that gets transformed to our problem) often can provide useful information to algorithm designers. Through this section we intend to show that proofs of NP-Completeness are not as un-interesting as the research community believes. Using an example we demonstrate that this is indeed the case.

We refer the reader to a paper by Murgai "On the Global Fanout Optimization Problem" [19]. The main contribution of this paper is the proof of NP-Completeness of the global fanout optimization (buffer insertion) problem. The author assumed the *load dependent delay model* to be valid. In this delay model each gate has a load sensitivity factor $?$, which is different for different input pins. The delay from an input pin *i* to output is given by

$$?(i) = ?(i) + ?(i) * Cout,$$

where $?(i)$, $?(i)$, and *Cout* are intrinsic delay, load sensitivity factor, and capacitive loading, respectively.

The proof of NP-Completeness transforms the 3SAT problem to an instance of the buffer insertion problem. In order to make a cogent explanation we outline some details of the transformation. Following is the decision problem associated with buffer insertion: *Given a combinational circuit with Pis and Pos and gates. The topology of the nets that connect the gates is fixed. Given the required time r for all primary outputs, a buffer b and a number D. Does there exist a buffering of nets such that the required time at each PI is at least D.*

The paper proves that 3SAT is satisfiable iff there exists a buffering of nets such that the required time constraints at all Pis are met. Each variable in the 3SAT problem was transformed to a circuit which had the structure shown in Figure 3.

Each clause also corresponds to a gate. Gate Bi in the variable circuit structure (shown above) has two kinds of input pins. If it appears in the positive phase in a clause, then we use the input pins on the left side of the gate else we use the ones on the right

side. If the net connected between gates Bi and Ai, is buffered then it corresponds to setting the truth value of the corresponding variable as TRUE else FALSE. Figure 3.1(A) shows that if that net is not buffered then the required time at certain input pins is larger than others. Similarly this behavior becomes reverse if that net is buffered, which is shown in the Figure 3.1 (B). Because of this phenomenon, if there was no truth assignment to 3SAT, there was no buffering solution to satisfy the primary input constraint. Similarly if there existed a solution to 3SAT, there existed a solution for the buffer insertion instance.
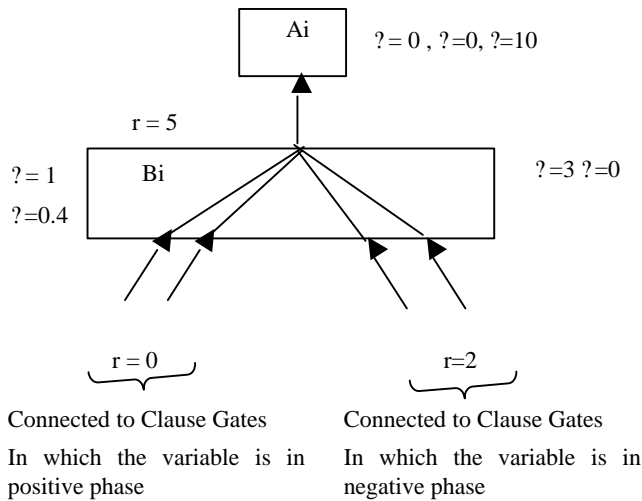

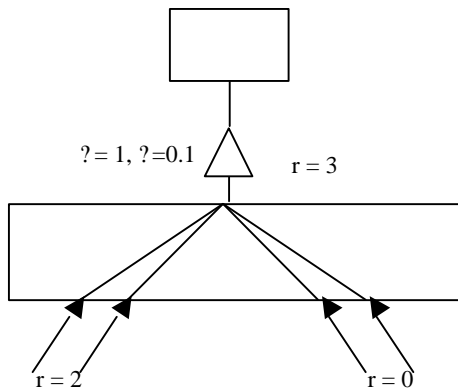**Figure 3.1(A)—Circuit Corresponding to Variable X$_i$**

Figure 3.1(B)—Circuit Corresponding to Variable X$_i$

A similar observation is also made by the author himself. If a buffered solution of the net connected to the output of a gate is optimal w.r.t a particular input pin, it may not be optimal w.r.t. other input pins. The reason for this is that each input pin has a different value for ?. Hence there are conflicting choices and the problem becomes NP-Complete. This conclusion was drawn directly from the transformation used in the proof of NP-Completeness. The buffering of net connected to gate Bi was beneficial for some input pins (in terms of required time) while for others it was not. The next question is whether we can use this information to say something about optimality of algorithms. The author presents sufficient conditions for which there exists an optimal algorithm to solve the problem. If the buffered solution of the net is such that it is the best solution for all the input pins, then the global problem is polynomially solvable (if the local problem is polynomially solvable). For every net, find a locally optimal buffering solution. Since this solution is optimal for all the input pins, the global algorithm can be optimally solved too. Such conclusions were made directly from the proof of NP-Completeness. The proof also suggests that sub-structures or sub-problems in which ? does not vary too much, the effectiveness of the algorithm is more, thus suggesting ways of partitioning the circuit.

The above discussion shows that proofs of NP-Completeness can point in useful directions. They can assist the algorithm designers in finding right ways of solving the problem.

4. PROOF OF A GREEDY ALGORITHM

In this section, we show the power of greedy algorithms through proof of correctness. As an example, we use the Prim's well-known algorithm for minimum spanning trees (MST) [10].

A *greedy algorithm* always makes a locally optimal choice in hope that this leads to a globally optimal solution. Of course, greedy algorithms do not always find a globally correct solution. But, it is quite powerful and works for a well for a wide variety of problems e.g. Dijkstra's shortest path algorithm and Prim's MST algorithm. A correct greedy algorithm is often the simplest, most elegant algorithm for the problem.

An optimization problem that can be greedily solved often exhibits two properties: optimal substructure and greedy choice. *Optimal substructure* means that an optimal solution to the problem contains optimal solutions to sub-problems. The *greedy choice property* means that a globally optimal solution can be built using a series of locally optimal choices. Matroid theory can be used to determine if an algorithm can be solved through a greedy method [11].

Prim developed a correct greedy algorithm for the minimum spanning tree problem. The algorithm iteratively grows a tree by greedily selecting edges to grow the tree. A safe edge, which adds minimum weight to the tree, is added at each step of the algorithm. The algorithm completes when all of the vertices are a part of the tree.

Prim's algorithm exhibits optimal substructure as the sub-tree grown over the course of the algorithm is an MST over the set of points that it spans at any instant. Also, it demonstrates the greedy choice property since it adds the edge, which at that instant minimizes the current spanning tree; it has no regard for the global implications of choosing an edge.

Prim's algorithm is useful for solving the NP-Complete problem of finding the Steiner Minimum Tree (SMT). Many SMT algorithms use Prim's algorithm as basic part of their method. Based on the relationship between MST and SMT (the 2/3 bound) all Steiner heuristics based on MST enjoy a good approximation

bound. Indeed, most effective SMT heuristics used in practiced are based on MSTs.

Correct greedy algorithms play an important role in solving NP-Complete problems. Examples, as mentioned abover, are Steiner tree heuristics. A number of other greedy algorithms have been developed and are being used in industry. The only drawback of these techniques is that they normally not analyzed well.

5. GREEDY ALGORITHM VS GLOBAL ANALYSIS

In this section we look at a popular greedy algorithm and show a more effective algorithm can be designed once we view the problem more globally.

Consider a *directed acyclic graph* $G = (V, E)$. A set of *sources* (or *primary inputs*) I of G is a set of vertices without incoming edges, and a set of *sinks* (or *primary outputs*) O is a set of vertices without outgoing edges. Given $|I|$ signals each starting from a source I_i at time $a(I_i)$, we consider their propagation towards O along directed edges and vertices in G, assuming that each vertex $v \in V$ is associated with a *delay* $d(v)$ which represents the time it takes for a signal to pass through v and that there is no delay on edges. The latest time of signals to arrive at the output of any vertex $v \in V \setminus I$ is recursively given by

$$a(v) = \max_{u \in FI(v)} (a(u) + d(v)), \qquad (1)$$

where $FI(v)$ is a set of (immediate) *predecessors* (or *fanins*) of v, and $a(v)$ is called *arrival time* of v. If signal at a sink O_j is required to arrive by time $r(O_j)$, the signal at output of any vertex $v \in V \setminus O$ is required to arrive by time

$$r(v) = \min_{w \in FO(v)} (r(w) - d(w)), \qquad (2)$$

where $FO(v)$ is a set of (immediate) *successors* (or *fanouts*) of v, and $r(v)$ is called *required time* of v. *Slack*, $s(v)$, for vertex v is the difference between $r(v)$ and $a(v)$, i.e., $s(v) = r(v) - a(v)$.

A *vector* of delays $D(V) = [d(v_1) \ d(v_2) \ \ldots \ d(v_n)]$ (where $n = |V|$) is called *delay distribution* of G. Similarly, a vector of slacks $S(V) = [s(v_1) \ s(v_2) \ \ldots \ s(v_n)]$ is called *slack distribution*, and $|S(V)| = \sum_{k=1}^{n} s(v_k)$ is called *total slack* (denoted by S_t). G is said to be (timing) *safe* if and only if $S(V) \geq 0$ (unless otherwise specified, a graph under consideration is initially safe. In fact, an unsafe graph can be made safe by, for example, increasing required time sinks). If an *incremental* delay $\Delta d(v) > 0$ is assigned to any vertex $v \in V$, the slack distribution will be reduced, depending upon the topology of G. In other words, we are assigning a *slack* to, or obtaining a *budget* from, the graph. In general, we define a *budget management* (or *slack assignment*) to be a vector of incremental delays $\Delta D(V) = [\Delta d(v_1) \ \Delta d(v_2) \ \ldots \ \Delta d(v_n)] > 0$ which updates delay distribution from $D(V)$ to $D_\Delta(V) = D(V) + \Delta D(V)$ and hence updates slack distribution from $S(V)$ to $S_\Delta(V)$ (The terms "budget management" and "slack assignment" will be used interchangeably throughout the paper). If G is still safe (i.e., $S_\Delta(V) \geq 0$), then the budget management is said to be *effective* and $|\Delta D(V)| = \sum_{k=1}^{n} \Delta d(v_k)$ is called *effective budget* on G. Note that the amount of possible effective budget management for G can be exponential function of its size n. An *optimal budget management* (denoted by $\Delta_m D(V)$) is an effective budget management which results in *maximum effective budget* $B_m = |\Delta_m D(V)|$. Figure 5.2. shows a directed acyclic graph and an optimal budget management thereof.

The well-known *zero-slack algorithm* (ZSA) for slack assignment has been proposed [1] with the goal of generating performance constraints for VLSI layout design. Some improved versions of ZSA can be found in [2, 3]. In [4], a *convex programming* problem formulation was given to tackle a *delay budgeting* problem, which is equivalent to slack assignment. While these prior works focus on applications in VLSI (very-large-scale integrated circuits) layout, the main disadvantage is that they are not able to provide an "optimal" budget management. In this section we analyze some characteristics of slack, present an optimal algorithm for budget management, and extend its applications to more general optimization domains.

First we review ZSA [1] briefly. The basic idea of ZSA is to start with vertices with minimum slack and locally perform slack assignment such that their slacks become zero. This process repeats until the slack of all vertices is zero. More specifically, at each iteration, ZSA identifies a path on which all vertices have minimum slack (denoted by s_{min}), then assigns to each vertex an incremental delay s_{min} / N_{min}, where N_{min} is the number of vertices on the path. In Figure 5.2., for example, path (v_1, v_3, v_4) is first found and each of these three vertices is assigned an incremental delay of 5/3. The slack of all vertices except v_2 in the figure becomes zero, while $s(v_2)$ is updated as 5/3. After assigning an incremental delay of 5/3 to 2 in second iteration, the slack distribution is updated to be zero, and the algorithm terminates with effective budget of 20/3 (note that maximum effective budget of Figure 5.2. is 10). This example shows that while ZSA is simple to use and easy to implement, the result is far from optimal budget management.

As opposed to ZSA, the global algorithm begins with a set of vertices V_m with maximum slack. Then we construct a transitive slack-sensitive graph $G_t(\Delta) = (V_m, E_m^t)$. Based on discussions in the above section, a maximum independent set (MIS) [6] of $G_t(\Delta)$ corresponds to a set of vertices $V_{MIS} \subseteq V_m$ such that the number of vertices which are slack-sensitive to any vertex $v \in V_{MIS}$ is minimized. We assign an incremental delay Δ to each vertex in V_{MIS}, and update the slacks of vertices in V_m. This process of constructing $G_t(\Delta)$, finding MIS and updating slacks continues until the slack distribution becomes zero. Clearly, each step of the process contributes $\Delta|V_{MIS}|$ to final effective budget. Finding an MIS of $G_t(\Delta)$ is straightforward. This can be done by identifying a vertex v_{min} with minimum *degree* in $G_t(\Delta)$ iteratively. Initially let

V_{MIS} be empty. We add v_{min} into V_{MIS} and update the graph by deleting v_{min} and all its neighbors (together with their associated edges) from the graph. This procedure continues until the graph becomes empty.

Maximum-Independent-Set based Algorithm (MISA)
/* An algorithm for finding budget management */
Input: graph $G = (V, E)$ and timing constraint T_{spec}
Output: budget management $?D(V)$
begin
Compute slack for each vertex $v ? V$, and find maximum slack s_m
    Initialize $?D(V) = 0$
    while ($s_m > 0$)
        Construct a transitive slack-sensitive graph
            $G_t(?)$ using $G_t(?)$-algorithm;
        Find maximum independent set V_{MIS} of $G_t(?)$
        Assign an incremental delay $?$ to each vertex in
        V_{MIS}, i.e., $?d(w) ? ?d(w) + ?$, $? w ? V_{MIS}$
        Update slack distribution and s_m
 end
Figure 5.1- Maximum-Independent-Set based Algorithm

$D(V) = [\ 0\ \ 0\ \ 0\ \ 3\ \ 2\ \ 3\ \ 2\ \ 0\ \ 0\ ]$, $?_mD(V) = [\ 0\ \ 0\ \ 3\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ ]$
$S(V) = [\ 0\ \ 0\ \ 4\ \ 0\ \ 1\ \ 0\ \ 2\ \ 0\ \ 2\ ]$, $S_t = |S(V)| = 9$ $B_m = |?_mD(V)| = 5$
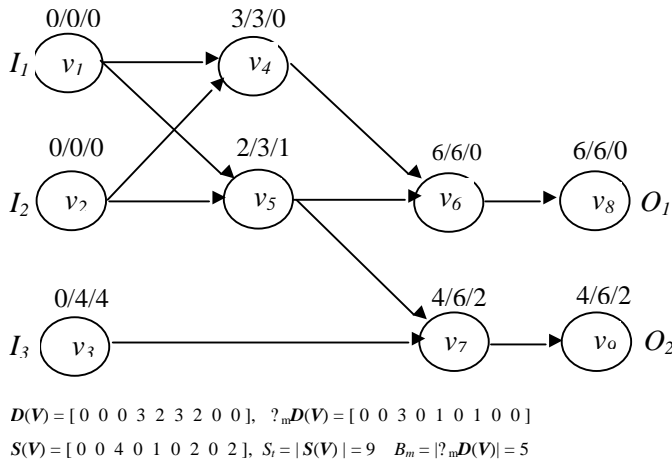
Figure 5.2- Optimal Budget Management

The pseudo code of our MIS-based algorithm for budget management is given in Figure 5.1. It has been proved [22] that MISA produces an optimal budget management, which results in maximum effective budget of **G**.

Two well-known applications of budget management are gate sizing and performance generation during placement.

6. APPROXIMATION ALGORITHMS
One method to solve an NP-hard problem is to develop a heuristic algorithm. The heuristic method should be able to produce a result close to optimal solution for the problem. As mentioned earlier showing a problem to be NP-hard is an important theoretical effort. However the problem needs to be solved. The goal is to find a feasible solution close to optimal solution that can be computed efficiently. Hence there is trade-off between optimality and tractability [17]. An $?$-*Approximation* algorithm for minimizing problem P produces a solution for any instance $I ? P$ that is at most $?$ times the optimum solution [17]. It is good to study the closeness and approximation of existing algorithms. Assume performance of an algorithm is observed only by running on a set of benchmark circuits. Therefore different algorithms for a given problem can be compared only with a set of samples. Also results cannot be a good guidance to improve the performance of the methods. We need to evaluate the quality of heuristic algorithms. Using analysis methods for approximation algorithms may result in deriving lower bounds or evaluating limits of approximability. Lower bounds on approximation algorithms help us know whether there is a better approximation solution. However, most of the times it is not easy to analyze the lower bounds of hard problems. In [17], there are examples of such problems for which no lower bounds have been obtained yet.

Many of heuristics for problems in VLSI CAD can be easily analyzed and theoretically understood. This may lead to ability to obtain an algorithm closer to optimum solution.

As an example we analyze the performance of a simple clustering algorithm proposed by Gonzalez[16]. Gonzalez proposed a $O(kn)$ algorithm for a clustering problem. k is the number of clusters and n is the number of nodes to be clustered. The algorithm is simple and can be easily analyzed. He has shown that the proposed heuristic gives a solution within two times the optimal solution.

The problem is formally stated as: Given an undirected weighted graph f, an objective function f, and integer k, partition the graph into k clusters such that the value of objective function corresponding to the partition is minimized. Clustering problem formulated above is NP-hard [16, 17]. This problem is studied well in detail in [16]. The graph is assumed to be a complete graph. The objective is minimizing the maximum weight of an edge that is inside a cluster. Also another important assumption is that weights of the edges in graph G satisfy the triangle inequality.

The algorithm works as follows: All the nodes are initially assigned into one cluster. One node is arbitrarily chosen as a head of the cluster. A node that has the longest distance from the head is moved to a new cluster. That node will be labeled as the head of the new cluster. Basically this algorithm tries to minimize the distance of each node inside a cluster to the head of the cluster. Therefore important task is to choose the heads for clusters. If there is a node whose distance from the head of the new cluster is less than its distance from the head of the cluster it belongs to, the node will be moved to the new cluster. At each stage i, node v_i that has the maximum distance from the head of its cluster b_i is moved to new cluster b_{i+1} as the head of cluster b_{i+1}. Any node in any other clusters that have shorter path to new head v_i compared to its distance from the head of the current cluster it belongs to will be moved to cluster b_{i+1}. Each stage generates one more

cluster. Therefore *k* clusters are constructed by repeating the same process for *k* times.

It is simple to show that the solution generated by this algorithm has an objective value within two times the value for optimum solution, *OPT(G)* [16]:

Lemma 6.1. *Algorithm proposed in [16] generates a solution with an objective function value ? 2? OPT(G).*

Proof. *Suppose node v_j in cluster b_j has the longest distance from the head of cluster b_j. We assume that distance is h. According to triangle inequality, the maximum distance of any two nodes in cluster b_j is at least 2h. Since node v_j never becomes a head of any new cluster, it means that the distance of node v_j to the head of any other cluster is ? h. Therefore the heads of clusters together with node v_j generate a (k+1)-clique of weight h. In this (k+1)-clique the distance of any two nodes is ? h. According to lemma 6.2, the objective value of optimal solution is at least h. This concludes that the objective function value of the current solution is at least 2h, hence two times the objective value for optimal solution.*

Lemma 6.2. *If there is a (k+1)-clique of weight h for G, the OPT(G)? h [16].*

In [16] it is also proven that computed approximation bound is the best possible approximation for such a clustering problem if P ? NP. Referring to [17], a polynomial ? -approximation algorithm is *best possible* if the problem of approximation within (? -?) is NP-Complete for any ?? 0. Gonzalez [16] has proven that the problem of approximation within 2-? is NP-Complete.

It is very important to analyze approximation algorithms. If the approximation derived for an algorithm is not sufficiently strong, we may instead use a simple method that has a strong guarantee. Then we can improve the simple algorithm by adding more local optimization techniques. However most of the times relatively strong approximations for algorithms can be obtained without much computational effort. The analysis for a simple clustering algorithm described in this section is such an example.

7. PROBABILISTIC ALGORITHMS
In this section, we gave an overview of probabilistic algorithms and discuss their merits. We examine a multi-way cut algorithm by Karger [6], which elegantly proves through probabilistic analysis many properties associated with his algorithm. (For n overview of probabilistic algorithms see [9].)

A *probabilistic algorithm* is, quite simply, algorithm that makes random choices during execution. It is also known as a random algorithm. With the use of probability theory, probabilistic algorithm can yield expected runtimes as well as bounds of the solution quality with uncertainty (hopefully small). Probabilistic algorithms often run faster than the corresponding deterministic algorithms. Also, many probabilistic algorithms are easier to implement and describe than deterministic algorithms of comparable performance [9].

Karger uses the simple graphical method of *contraction* to create a powerful clustering algorithm. Given two vertices v_1 and v_2, remove v_1 and v_2, replacing them with a new vertex v. The set of edges incident on v are the union of the set of edges incident to v_1 and v_2. Edges from v_1 and v_2 with mutual endpoints, e.g. $e_1 = (v_1, x)$ and $e_2 = (v_2, x)$, may or may not be merged into one edge; Karger does not merge them. Karger removes any edges between v_1 and v_2 to eliminate self-loops. The algorithm that Karger uses for clustering is shown in Figure 7.1.

As you can see, the algorithm is very simple. Arguably, the "simplest" deterministic partitioning (essentially equivalent to

---

Contraction algorithm for clustering

Input: directed acyclic graph $G = (V, E)$, number of partitions k
Output: k sets of vertices
begin
    while |V| > k
    do choose and edge *e(u,v)* at random
        contract both *u* and *v*
end

Figure 7.1 – Karger's contraction algorithm for clustering

---

clustering when $k = 2$) algorithms are the well known KL [8] and FM [7] algorithms. Obviously, the probabilistic algorithm is easier to describe and implement compared to FM or KL.

The most intriguing property of probabilistic algorithms is the ability to use the "simple" algorithm to prove powerful properties related to both the algorithm and the problem. Continuing with our example, Karger proves the following:

Corollary 7.1: *If we perform* $O(n^2 \log n)$ *independent contractions to two vertices, we find a min-cut with high probability. In fact, with high probability we find* every *min-cut.*

Theorem 7.1: *Stopping the Contraction Algorithm when r vertices remain yields a particular minimum r-way cut with probability at least*

$$r\binom{?}{?}\binom{n}{r?1?}^{?1}\binom{?n?1?}{?r?1?}^{?1}$$

Proof: *The key to the analysis is bounding the probability* p *that a randomly selected graph edge is from a particular minimal r-cut. Suppose we choose* r *–1 vertices uniformly at random, and consider the r-cut defined by taking each of the vertices as one member of the cut and all the other vertices as the last member. Let* f *be the number of edges cut by this random partition, and* m *the number of the graphs edges. The number of edges we expect to cut is*

$$E?f?? \; {?\atop?}1 ? {?\atop?}1 ? \frac{r?1}{n}?? {??\atop??}1 ? \frac{r?1}{n?1}?? {??\atop??}m,$$

since the quantify in brackets is just the probability that a single edge is cute. Since f *can be no less than the value of the minimal r-cut,* E[f] *must also be no less than the min-cut. We can therefore deduce that the probability that a particular minimum r-cut survives the reductions process until there are* r *vertices remaining is at least*

$$\prod_{u?r?1}^{n} ?1 ? \frac{r?1}{u}?? 1 ? \frac{r?1}{u?1}?$$

$$? ? \prod_{u?r?1}^{n} ?1 ? \frac{r?1}{u}? ? \prod_{u?r?1}^{n} ?1 ? \frac{r?1}{u?1}?$$

$$? r \binom{n}{r?1}^{?1} \binom{n?1}{r?1}^{?1}$$

The previous corollary and theorem give probabilistic properties related to the algorithm itself. Unlike most existing partitioning/clustering algorithms, Theorem 7.1 gives bound on the solution quality. Since benchmarks are at a premium in the CAD community, bounds on solution quality are particularly important. It is often the case, that papers "prove" their algorithms to be better than existing algorithms based solely on small amount of benchmarks (which are often outdated). While there is huge need for better benchmarking – which is another subject altogether – in the community, giving solution quality bounds can not only be used as definitive proof for algorithm supremacy, it can also gave great insight into the problem itself. Consider the following corollaries:

Corollary 7.2: *The number of minimum r-cuts of a graph is no more than $O(n^{2(r-1)})$.*

Corollary 7.3: *The number of r-cuts within a factor of k of the optimum is $O(n^{2k(r-1)})$.*

Both are a consequence of Theorem 7.1. And both give tremendous insight into the clustering problem.

Probabilistic algorithms are often simple to derive and implement. Yet, through the use of probability theory, they provide a powerful mechanism for provide solution quality bounds. Furthermore, they allow identification of some fundamental properties of the problem.

8. CONCLUSION
In this paper, we looked into the algorithmic and analysis aspects of some fundamental CAD problems. We argued that careful analysis of algorithms is essential for determining the best approach to the problem. Specifically, we discussed the use of problem transformation to solve problems with exhibit similar nature. We discussed a method of problem transformation by extending 2-way partitioning algorithms to solve the k-way partitioning problem.

We also discussed the implication of lower bound problem transformation. We assert that proofs for NP-Completeness not only show the complexity of the problem, but also give tremendous insight into the computational bottleneck of the problem. This allows us to produce better heuristics to solve the problem.

Although there are effective (and at-times optimal) greedy algorithms there are many cases that an algorithm with a more global view produces much better results. Although global algorithms are normally slower than greedy techniques, they can shed new lights on the complexity of physical design problems. Greedy algorithms are often the simplest, most eloquent algorithms for solving a problem. Therefore, analysis of computational complexity and solution quality of greedy algorithms is straightforward compared to other types of algorithms.

Approximation algorithms give a bound on their solution quality. Analyzing and theoretically understanding approximation algorithms helps us understand the problems and improve the performance of the existing methods.

We discussed Karger's probabilistic algorithm for clustering. The algorithm uses probabilistic theory to show that it is correct with a small uncertainty. Perhaps more importantly, the algorithm gave insight into the problem. Karger was able to prove properties about the problem using his algorithm and probability theory.

9. REFERENCES

[1] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Transactions on Computer-Aided Design*, *CAD-8(8)*, pp.860-874, August 1989.

[2] T. Gao, P. M. Vaidya, and C. L. Liu, "A New Performance Driven Placement Algorihm," in *Proceedings of International Conference on Computer-Aided Design*, pp.44-47, 1991.

[3] H. Youssef and E. Shragowitz, "Timing Constraints for Correct Performance," in *Proceedings of International Conference on Computer-Aided Design*, pp.24-27, 1990.

[4] G. E. Tellez, D. A. Knol, and M. Sarrafzadeh, "A Graph-Based Delay Budgeting Algorithm for Large Scale Timing-Driven Placement Problems," in *Proceedings of fifth ACM/SIGDA Physical Design Workshop*, pp.234-240, April 1996.

[5] R. H. Moehring, "Graphs and Orders: the Roles of Graphs in the Theory of Ordered Sets and Its

Applications", *Published by D. Reidel Publishing Company, Edited by I. Rival, New York and London*, pp.41-101, May 1984.

[6] D. R. Karger, "Global Min-cuts in *RNC*, and Other Ramifications of a Simple Min-cut Algorithm, in *Proceedings of the ACM-SIAM Symposium on Discrete algorithms*, 1993.

[7] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitioning", in *Proceedings of the Design Automation Conference,* pp 175-181, 1982.

[8] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", in *Bell Systems Technical Journal,* 49(2): 291-37, 1970.

[9] R. Motwani and P. Raghavan, "Randomized Algorithms", *Published by Cambridge University Press, Cambridge UK and New York,* 1995.

[10] R. C. Prim, "Shortest Connection Networks and Some Generalizations", in *Bell System Technical Journal,* 36(6):1389-1401, 1957.

[11] B. Korte and L. Lovasz, "Mathematical Structures Underlying Greedy Algorithms", in *Fundamentals of Computation Theory,* 1981.

[12] C. K. Cheng and Y. C. A. Wei, "An Improved Two-way Partitioning Algorithm with Stable Performance", in *IEEE Transactions on Computer Aided Design*, 10(12): 1502-1511, 1991.

[13] L. Hagen and A. B. Kahng, "Fast Spectral methods for Ratio Cut Partitioning and Clustering", in *IEEE/ACM International Conference on Computer-Aided Design*, 1991.

[14] P. R. Suaris and G. Kedem, "Quadrisection: A New Approach to Standard Cell Layout", in *IEEE/ACM Design Automation Conference*, pp. 474-477, 1987.

[15] P. R. Suaris and G. Kedem, "Standard Cell Placement by Quadrisection", in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 612-615, 1987.

[16] T. F. Gonzalez, "Clustering to Minimize the Maximum Intercluster Distance", in *Theoretical Computer Science*, 38:293-306, 1985.

[17] D. S. Hochbaum, "Approximation Algorithms for NP-Hard Problems", *PWS Publishing Company,* 1995.

[18] M. R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to he Theory of NP-Completeness", W.H. Freeman and Company-1979.

[19] R. Murgai, "On the Global Fanout Optimization Problem", in International Conference on Computer Aided Design, Nov 1999, pages 511-515

[20] S. A. Cook, "The Complexity of Theorem-proving Procedures", in *Proceedings of 3rd Anniversary of ACM Symposium on Theory of Computing*, pp 151-158, 1971.

[21] R. M. Karp, " Reducibility Among Combinatorial Problems", in *R. E. Miller and J. W. Thatcher, Complexity of Computer Computations, Plenum Press*, pp. 85-103, 1972.

[22] C. Chen and M. Sarrafzadeh, "Potential Slack: An Effective Metric of Combinational Circuit Performance", in *ACM/IEEE International Conference on Computer-Aided Design*, 2000.

[23] E. F. Moore, "Shortest Path Through a Maze", in *Kluwer Proceedings of the International Symposium on Switching Circuits*, 1959.

[24] C. Y. Lee, "An Algorithm for Path Connection and its Applications", in *IRE Transactions on Electronic Computers*, 1961.