A Class of Bounded Approximation Algorithms for Graph Partitioning

Thomas A. Feo
Operations Research Group, Department of Mechanical Engineering, University of Texas, Austin, Texas 78712

Mallek Khellaf
Department of Industrial Engineering and Operations Research, University of California, Berkeley, California 94720

This paper considers the problem of partitioning the nodes of a weighted graph into k disjoint subsets of bounded size, such that the sum of the weights of the edges whose end vertices belong to the same subset is maximized. A class of approximation algorithms based on matching is presented. These algorithms are shown to yield practical worst-case bounds when k is large. Extensive empirical experimentation indicates that the methods produce consistently good solutions to an important VLSI design problem in a fraction of the time required by competing methods.

1. INTRODUCTION

The problem that consists of finding a partition of the nodes of an undirected graph in order to optimize a given measure is referred to as the graph partitioning problem. This clustering of the nodes of a graph into subsets or groups is shared by many combinatorial problems. The minimum cut problem is a simple example. The relationship between this problem and the well-known maximum flow problem was first shown by Ford and Fulkerson [8]. The minimum cut problem generalizes in a number of directions. For instance, if the number of desired subsets is k ($k > 2$), the problem is referred to as k-way graph partitioning [13]. Furthermore, constraints on the cluster size can also be imposed. This version is called minimum cut into bounded sets [10]. The most restrictive of these problems is when the size of the clusters, m, is specified. This is referred to as k-partition or m-dimensional matching on a graph, where the number of vertices in the graph is km. The case with $m = 2$ is the well-known weighted matching problem.

CCC 0028-3045/90/020181-015$04.00

It is important to note that partitioning the nodes of a graph, such that one minimizes the sum of the weights of the edges having incident vertices not in the same subset, is identical to clustering the nodes of the graph such that one maximizes the sum of the weights of the edges having incident vertices in the same subset. This follows from the fact that the sum of the weights of all edges in the graph is a constant and that each edge is either in a cluster or runs between two clusters.

Graph partitioning serves as an abstraction of several important problems. For instance, an application arises in Very Large Scale Integration (VLSI), the process by which a complex electronic circuit is implemented on a silicon wafer to form a chip. The circuit is often composed of many modules that communicate with each other through physical connections, i.e., wires. The first stage of integrating the modules is to physically place them relative to each other. The second stage is to globally route the wires connecting the modules. There are two basic objectives sought during the placement and routing stages of VLSI design. The first is to minimize total wire length, and the second is to minimize the area of the silicon wafer required for the chip. It is intuitive that grouping highly connected modules will facilitate the routing of the electrical connections and will reduce the layout area. If modules are modeled as nodes and the electrical connections between them by edges, the grouping of modules according to their connectivity is a graph partitioning (or clustering) problem [5, 14, 20].

Partitioning also arises when storing large computer programs onto paged memory. Programs are usually made up of a set of interconnected subroutines where the connections represent flow or transfer of control. The problem is to assign the subprograms to pages of a given size so as to minimize the number of references between objects that are stored on different pages. The size of each subroutine is fixed, and no subroutine may reside on two different pages. The graph-theoretic abstraction of this problem is that each subprogram is represented by a weighted vertex, where the weight is proportional to the size of the routine. A weighted edge is defined between each pair of routines (nodes) that communicate, where the weight is proportional to the frequency of access during the execution of the program. A desirable assignment of subprograms to pages is equivalent to a partitioning of the nodes of the graph such that the total weight of the arcs across the partitions is minimized and the weight of the nodes in each partition does not exceed the page capacity [18, 20].

A third application arises in group technology classification problems. Given a set of parts to be manufactured and the machining requirements for each, the objective is to classify the parts into families such that the processing requirements among parts in the same family are very similar. This problem is one of the primary steps in planning a flexible manufacturing system [4, 16, 19]. The parts are modeled as nodes of a complete graph with edge weights representing the similarity of the processing requirements of the incident vertices. Setting bounds on the size of the families completes the identification of this problem with the graph clustering problem.

Most of the research on graph partitioning has focused on variations in which the number of clusters is very small (i.e., 2 or 3). See Feo and Khellaf for a survey on graph partitioning [7]. Notice that in the three industrial applications mentioned the number of clusters is often large ($O(/V/)$), and that only a few attempts have been made to study and offer methods for these problems [2, 12]. The algorithm of Barnes involves computing eigenvectors, solving transportation problems, and performing a 2-exchange procedure. Barnes presents his method with respect to 2-partition and gives small numerical examples. However, the practicality of his procedure for general k-partitioning in terms of both running time and solution value is not discussed. Jackson and Kuh [12] propose a probabilistic exchange procedure. Their method is shown to produce good solutions but the CPU time required is excessive.

In this paper, we address the lack of practical heuristics for k-partitioning where k is large. The existence of an efficient optimal algorithm for this problem is unlikely in view of the NP-completeness proof we offer in the next section. The heuristics that we propose are based on grouping nodes into sets of two. These 2-node groups form the nodes of a new graph to which the grouping procedure is reapplied. The process is repeated until feasible-sized clusters are obtained. The grouping method at the basis of this class of heuristics is matching. Analysis of these heuristics shows that worst-case performance is bounded. Their practicality is also demonstrated via empirical experimentation on a variety of industrial as well as randomly generated VLSI problems.

2. TRANSFORMATIONS AND COMPLEXITY

This section presents two simple NP-completeness proof for k-partitioning. The first focuses on general edge weights, and the second assumes that the arc weights possess the triangle inequality property. For notational simplicity, the proofs are given in terms of m-dimensional matching on a graph (mDM) instead of the k-partitioning version of this problem. The formal definition of mDM follows.

Instance: Graph $G = (V, E)$, $|V| = km$, $m \geqslant 3$, a weight w_e for each edge e of E and an integer j.

Question: Is there a partition of V into disjoint sets $V_1, V_2, \ldots, V_k$, of m vertices each such that the sum of the weights of the edges that have both endpoints in the same set V_i (i.e., not cross edges) is greater than j?

Lemma 1. mDM is NP-complete.

Proof. The problem is in NP. The reduction is from graph partitioning [9]. Given a graph $G = (V, E)$ and integers $m \geqslant 3$ and j, find a partition of V into disjoint set $V_1, \ldots, V_k$ (k not fixed) such that the cardinality of each set is less than or equal to m, and the total number of cross edges is less than j. Let $n = |V|$. Consider the graph $G' = (V', E')$, where $V' = V \cup \{u_1, \ldots, u_{n(m-1)}\}$, and $E' = E$; that is, we add $n(m - 1)$ isolated vertices to V. Let $j' = j$, and $m' = m$. $G' = (V', E'), j'$ and m' define an instance of mDM where all edge weights are equal. There exists a partition of V into k disjoint sets, $V_1, \ldots, V_k$, such that $|V_i| \leqslant m$ for all $i = 1, \ldots, k$, and the number of cross edges is less than j, if there exists a partition of V' into n sets of m' vertices each such that the total number of cross edges is less than j'.

■

The above transformation is for unweighted graphs. The NP-completeness of the weighted case follows since it contains the unweighted instances as a special case. The transformation used in the proof is invariant to planarity, maximum vertex degree, or any other edge property. This implies that if graph partitioning is NP-complete for some edge property, then mDM is NP-complete even if the graph satisfies that property. A similar procedure can be used to transform the k-way graph partitioning problem to mDM (k fixed, and each subset containing no more than m nodes). It suffices to add $km - |V|$ isolated nodes to the original set of nodes.

Let A be an $n \times n$, real, symmetric matrix such that $a_{ii} = 0$ for all i. A is said to satisfy triangle inequality if $a_{ij} \leqslant a_{ik} + a_{kj}$ for all $1 \leqslant i, j, k \leqslant n$. Define ΔmDM as m-dimensional matching on a graph such that its edge weight matrix satisfies triangle inequality.

Lemma 2. ΔmDM is NP-complete.

Proof. The problem is in NP. The reduction is from mDM. Consider P an instance of mDM in which all edge weights are one or zero. With loss of generality, P can be defined on a complete graph $G = (V, E)$. Add a weight of one to every edge in E. The resulting problem, P', is an instance of ΔmDM. It has all edge weights equal to one or two and, thus, satisfies the triangle inequality property. Since both P and P' are defined on complete graphs, a feasible solution to either problem will possess exactly $b = |V|(m-1)/2$ edges inside the clusters. Thus, any feasible partition of V having weight j in P will have weight $j + b$ in P'. Likewise, any feasible partition of V having weight $j + b$ in P' will have weight j in P. ■

3. BOUNDED APPROXIMATION ALGORITHMS

Consider the k-partition problem where the desired number of nodes per cluster is m. With no loss of generality, assume that $|V| = km$, where k is a positive integer (add up to $m - 1$ isolated vertices if $|V|$ is not a multiple of m). Also assume that the graph is complete with zero weight edges added to insure this condition. A perfect matching will exist, but some of the matched edges may be of zero weight. Any feasible solution is a set of k clusters of $m(m-1)/2$ edges each. Procedures P1 and P2, described below, partition the nodes of this graph into k sets of m nodes each. P1 is for the case where m is even, whereas P2 is for m odd.

Procedure P1

```
Input.  A complete graph $G$, $|V| = km$, $m$ even, and a nonnegative weight $w_e$ for
        each edge $e$.
Step 1. Find a maximum weight perfect matching $M$ in $G$.
Step 2. do $i = 1, k$
            Choose arbitrarily a set, $\pi$, of $m/2$ nonselected edges of $M$.
            Mark the edges in $\pi$ selected.
            Place in cluster $i$ the nodes incident to the edges in $\pi$.
        od.
```

Procedure P2

```
Input.  A complete graph $G$, $|V| = km$, $m$ odd, and a nonnegative weight $w_e$ for
        each edge $e$.
Step 1. Find a maximum weight matching $M$ in $G$ containing $(m-1)|V|/2m$
        edges.
Step 2. do $i = 1, k$
            Choose arbitrarily a set, $\pi$, of $(m-1)/2$ nonselected edges of $M$.
            Mark the edges in $\pi$ selected.
            Place in cluster $i$ the nodes incident to the edges in $\pi$, and any one
              node not incident to $M$ which is not already included in a cluster.
        od.
```

In procedure P2, a maximum weight matching containing a specified number of edges is found. This can be accomplished in $O(|V|^3)$ time by using a maximum weight perfect matching algorithm [22]. Construct the graph $G' = (V', E')$, where $V' = V \cup U, U = \{u_1, u_2, \dots, u_q\}$, $E' = E \cup F$, and F is the set of all edges having their endpoints between U and V. To each edge of F is assigned a large weight w. Let M' be a maximum weight matching of G'. $M' = E_1 \cup F_1$, where $E_1 = M' \cap E$, and $F_1 = M' \cap F$. If the weight w is sufficiently large (i.e., larger than the sum of all original edge weights), then M' contains exactly q edges of F. Moreover, $E_1 = M' \cap E$ is a maximum weight matching of G containing $(|V| - q)/2$ edges.

Theorem 1. If edge weights satisfy triangle inequality, the value of an optimal solution of the m-dimensional matching problem is at most $2(m-1)/m$ times the value of the solution produced by procedure P1. The bound is $(m-1)$ in the general case.

Proof. Let G be a complete edge weighted graph containing n (even) vertices and let M^* be a maximum weight perfect matching in G. Define $w(M^*)$ to be the weights of the edges in M^* and $w(G)$ the total edge weight in the graph. Then, $w(G) \leqslant (n-1)w(M^*)$, since the edges of a complete graph of n nodes can be partitioned into $(n-1)$ perfect matchings, and M^* is a maximum weight matching.

Define $w(P1)$ to be the weight of the solution produced by procedure P1, and let M be the maximum weight perfect matching obtained in Step 1. Define $w(\text{Opt})$ to be the weight of an optimal solution to the partitioning problem. Let C_i^* be the ith cluster in an optimal solution, and C_i be the ith cluster in the heuristic solution. Both C_i^* and C_i are complete graphs of m nodes each.

$$w(\text{Opt}) = \sum_{i=1,k} w(C_i^*), \text{ and } w(P1) = \sum_{i=1,k} w(C_i).$$

Define M_i^* to be a maximum weight matching in C_i^*. $w(C_i^*) \leqslant (m-1)w(M_i^*)$ holds in view of the first paragraph of this proof. Let $M^* = \{\cup M_i^* : i = 1, \dots, k\}$. Thus, $w(\text{Opt}) \leqslant (m-1)w(M^*) \leqslant (m-1)w(M)$. The last inequality holds because M is a maximum weight matching in G.

Denote M_i to be the set of edges of M in C_i. $|M_i| = m/2$, and $w(C_i) = w(M_i) + w(R_i)$, where R_i is the set of edges in C_i but not in M. Suppose that the nodes of C_i are numbered $1, 2, \ldots, m$. Denote edge (i, j) by e_{ij}. With no loss of generality, assume that $M_i = \{e_{12}, e_{34}, \ldots, e_{m-1m}\}$. Then, $w(e_{12}) \leqslant w(e_{1r}) + w(e_{r2})$ for all $r \neq 1, 2$, because of the triangle inequality property. Hence,

$$(m-2)w(e_{12}) \leqslant \sum_{r=3,m} w(e_{1r}) + w(e_{r2}), \text{ and}$$

$$(m-2)\sum_{e_{ij}\varepsilon M_i} w(e_{ij}) = (m-2)w(M_i) \leqslant \sum_{e_{ij}\varepsilon M_i}\sum_{r=1,m} w(e_{ir}) + w(e_{rj}) = 2w(R_i).$$

Thus, $w(C_i) \geqslant (m/2)w(M_i)$, which implies that $w(P1) \geqslant (m/2)w(M)$. Therefore,

$$w(\text{Optimal}) \leqslant [(m-1)w(M)]/[(m/2)w(M)] = 2(m-1)/m.$$

We can conclude that $w(P1) \geqslant w(M)$ when triangle inequality does not hold, and the bound becomes $m - 1$. ■

Theorem 2. The value of an optimal solution to a triangle inequality m-dimensional matching problem is at most $2m/(m + 1)$ times the solution produced by procedure $P2$, and at most m times the value of the heuristic solution in the general case.

Proof. Let G be a complete edge weighted graph containing n (odd) vertices, and define M^* to be a maximum weighted matching in G containing $(n - 1)/2$ edges (i.e., all nodes are matched except one). Assume with no loss of generality that node v is the unmatched vertex. The complete graph on the set of nodes $1, 2, \ldots, n - 1$ can be partitioned into $(n - 2)$ perfect matchings. Define E' to be the set of edges having node v as an incident vertex. We have $w(G) \leqslant (n-2)w(M^*) + w(E')$. Denote e_{ij} to be an edge of M^*. Then, $w(e_{ij}) \geqslant w(e_{ik})$, and $w(e_{ij}) \geqslant w(e_{jk})$, because M^* is a matching of maximum weight. Thus, $w(E) \leqslant 2w(M^*)$, and $w(G) \leqslant nw(M^*)$.

Define C_i to be the ith cluster of the heuristic solution. Suppose that the nodes of C_i are numbered $1, 2, \ldots, m$ and the node v is not incident to an edge of M. $C_i = M_i \cup R_i \cup E_i$, where M_i is the set of edges of C_i that intersect M, E_i is the set of edges running between node v and any other node of the cluster, and R_i is the set of edges of C_i not in M_i or E_i. Repeating the argument of the proof to Theorem 1, we have: $2w(R_i) \geqslant (m-3)w(M_i)$, and $w(E_i) \geqslant w(M_i)$. Both inequalities are due to the triangle inequality property. Thus, $w(C_i) \geqslant (m+1)w(M_i)/2$. This implies $w(P2) \geqslant (m+1)w(M)/2$, and a bound of $2m/(m + 1)$. If the graph does not satisfy triangle inequality, it is not possible to conclude that $w(P2) \geqslant (m+1)w(M)/2$, and the bound is therefore m. ■

4. EMPIRICAL RESULTS

This section presents three sets of empirical results. The first is obtained from experiments performed on 10 industrial VLSI partitioning problems in which the cluster size is restricted to five elements or less. The second set is based on five variations of each of the industrial instances in which the cluster size is restricted to four vertices or less. In the third set, we present results on five small randomly generated problems for which exact solutions are obtainable via a branch-and-bound procedure employing a Lagrangian relaxation technique.

Developing a VLSI circuit often requires the aggregation of newly and previously designed electronic modules. A common method of synthesis is to start with an initial placement of the modules comprising the circuit, find a global routing of the interconnections between the modules, and attempt a detailed routing. An infeasible outcome of the detailed routing leads to revisions in the placement and global routing. This procedure, although it yields reasonable solutions, is extremely time consuming as it often requires reworking the placement and global routing many times. Note that such an approach is not compatible in a real-time computer-aided design environment in which small design specification changes can lead to excessive user waiting times.

Dai and Kuh [6] have recently devised a new and successful strategy for placement and routing (Hierarchical Floor Planning Approach). Their method is based on clustering together those modules that are highly interconnected. A complete graph is defined with its nodes representing the modules and edge weights proportional to the number of electrical interconnections between the incident nodes. In the first iteration of their method, the nodes (modules) of the graph are clustered into groups of 4 or 5, such that the sum of the weights of the edges in all clusters is maximized. In subsequent iterations, the clusters themselves are combined into groups of 4 or 5. The iterations continue until there is only one cluster. This iterative procedure produces a hierarchical cluster tree. Within each iteration, a graph partitioning problem is solved.

The hierarchical tree is the input to a floor planner (placement procedure) and global router. The floor planner and global router perform a top-down traversal of the hierarchy. At each level, they simultaneously perform an exhaustive enumeration of possible layouts and global routings. This exhaustive search is possible because of the small number of elements in each cluster (five or less). The advantage of this hierarchical approach is that a relative placement and global routing are obtained that are not subject to revision by detailed routing considerations. Thus, the approach does not have to backtrack and is efficient.

It is important to note that the results obtained by the placement and routing procedure is directly influenced by the ability of the clustering method employed. Dai and Kuh state that their design system will possess both state-of-the-art speed and performance given a fast and tightly bounded heuristic for clustering [6]. An informal competition took place among faculty, researchers, and graduate students at Berkeley. Our double-matching method was selected as the best overall performer with respect to solution value and speed. It has been integrated into Berkeley's latest and most successful computer-aided VLSI design system (BEARS).

Table Ia gives the dimensions of the input graphs representing the 10 industrial problems considered. The number of nodes and density (the percentage of arcs in the complete graph of nonzero weight) is indicated for each problem. *Dummy nodes* corresponds to the number of isolated vertices added to the graph so that its total number of vertices becomes a multiple of four. *Total edge weight* is the sum of the weights of all the arcs in the graph. For this first set of 10 problems, an upper bound of 5 vertices in each cluster is imposed.

TABLE I. Graph characteristics representing 10 industrial problems and their solution values and times.

a. Graph characteristics

Instance	No. nodes	Dummy nodes	Density (%)	Total edge weight
RAL 12	12	0	69	590
BARNS	20	0	28	550
RAL 30	32	2	31	1040
RAL 33	36	3	70	820
AMI 33	36	3	71	900
AMI 49	52	3	55	4080
RAL 49	52	3	32	3770
INTEL	64	2	66	720
RAL 101	104	3	10	2360
RAL 122	124	2	8	2630

b. Solution values and times

	Heuristic 1		Heuristic 2		Heuristic 3		Lookahead		Annealing	
Instance	Value	Sec	Value	Sec	Value	Sec	Value	Sec	Value	Sec
RAL 12	449	0.00	449	0.00	449	0.00	449	0.3	449	5.5
BARNS	270	0.01	270	0.01	270	0.01	270	0.1	270	8.4
RAL 30	385	0.03	423	0.02	416	0.03	415	7.4	412	16.6
RAL 33	645	0.05	645	0.02	650	0.05	651	5.4	651	15.0
AMI 33	643	0.05	650	0.02	651	0.05	643	13.9	656	14.3
AMI 49	1211	0.14	1231	0.08	1211	0.13	1240	16.0	1220	17.8
RAL 49	2552	0.13	2591	0.07	2585	0.13	2420	19.8	2540	23.8
INTEL	674	0.22	674	0.13	674	0.15	674	3.1	674	27.1
RAL 101	773	0.93	755	0.53	751	0.63	728	105.0	778	109.1
RAL 122	933	1.56	905	0.87	916	1.34	952	89.1	1000	85.0

Three variations of the matching-based heuristic were implemented and are compared to two probabilistic exchange procedures [12]. Other procedures that competed are not discussed because of their poor solution value performance. Our heuristics apply matching to find a clustering of the nodes of the graph into sets of four elements each. Postprocessors, employing exchange procedures, then allow expansion of each cluster to five elements.

Heuristic 1

Input. A complete graph G, $|V| = 4k$, and a weight w_e for each edge e.
Step 1. Find a maximum weight perfect matching M_1 of G.
Step 2. Construct G' by contracting all the edges of M_1, and combine resulting parallel edges by summing their edge weights.
Step 3. Find a maximum weight perfect matching M_2 of G'.

Step 4. Form the k disjoint clusters of 4 vertices each by contracting all the edges of M_2.
Output. k clusters of V of 4 vertices each.

The complexity of Heuristic 1 is dominated by Step 1. The time required to find a maximum weight matching is $O(|V|^3)$ [23]. The construction of G' at Step 2 requires summing the weights of $|V| - 2$ edges per node in the graph—hence, a total of $O(|V|^2)$ operations. The complexity of Step 3 is $O((|V|/2)^3)$, and the complexity of Step 4 is $O((|V|/2)^2)$. The running time of Heuristic 1 is therefore $O(|V|^3)$. In view of Theorem 1, Heuristic 1 has a worst-case bound of three.

Although polynomial, an $O(|V|^3)$ implementation of Heuristic 1 could be computationally expensive for large-scale problems. Heuristics for weighted matching can be used to compute suboptimal matchings instead of maximum weight matchings in Steps 1 and 3 of Heuristic 1. However, the final solutions produced are observed to be inferior when such procedures are used. Two suboptimal matching heuristics were tested.

The first referred to as GREEDY [1] consists of picking a vertex i at random and choosing the edge (i,j) adjacent to i of maximum weight. The two end vertices of (i,j) are removed along with all edges adjacent to both i and j. The procedure terminates when no vertices are left. The complexity of this procedure is $O(|V|^2 \log |V|)$. Let **Heuristic 2** be the procedure obtained by finding a maximal weight matching using GREEDY at Steps 1 and 3 of Heuristic 1. Note that GREEDY has a worst-case bound of two [1]. The proof of Theorem 1 can be modified to show that Heuristic 2 has a worst-case bound of six for general edge weights and of three in the triangle inequality case.

A second matching heuristic, called PARDIV, consists of dividing the nodes of the graph into q equal parts so that $|V|/q$ is even. The idea behind this heuristic is to solve q matching problems of small or moderate size to optimality instead of one large matching problem. This scheme, used to solve various combinatorial problems, reduces the computation time needed to generate a solution at the expense of the quality of the solution. Let **Heuristic 3** be the procedure obtained by replacing Step 1 of Heuristic 1 by "Find a perfect matching M_1 of G using PARDIV." The complexity of Heuristic 3 improves inversely proportional to the magnitude of q, whereas intuitively, the quality of the solution may be proportional to q.

In our first set of experiments, the solutions generated by the above heuristics are submitted to a postprocessor routine that allows the size of the clusters to increase to five elements each.

Postprocessor 1

```
Input.   A feasible partition of the nodes of the given graph.
Step 0.  Set the incumbent solution equal to the given partition.
Step 1.  do $i = 1, |V|$
           do $j = 1, k$
             if (feasible and the objective value improves)
               Move node $i$ in the incumbent partition to cluster $j$.
```

```
            fi
          od
        od
```
Output. A feasible partition of the nodes of the input graph.

This simple exchange procedure considers every nodes of the graph and moves it from its current cluster to all other clusters. The move is accepted if it yields a better solution. The number of iterations of this procedure is $|V|k$ since each node is moved once to every cluster. The update of the cost at each iteration can be performed in O(m) time, where m is the number of elements in a cluster. The complexity of the procedure is $O(|V|^2)$.

The solution produced by Postprocessor 1 is submitted to a simple two-exchange method, **Postprocessor 2**. This procedure considers every edge of the complete graph and performs a swapping of its end vertices. The solution is recorded if it is better than the incumbent solution. The complexity is $O(m|V|^2)$ since the procedure goes through $|V|^2$ iterations, each requiring O(m) time.

Two probabilistic optimization methods also competed. The first known as Simulated Annealing is a special case of a wider class of adaptive heuristics for combinatorial optimization. Simulated Annealing was proposed by Kirkpatrick et al. [17]. The method is often seen as an extension of a Monte Carlo method developed by Metropolis et al. [21] to determine the equilibrium of a collection of atoms at a given temperature.

Jackson and Kuh [12] proposes another simple adaptive heuristic for combinatorial optimization. His method, called N Step Lookahead, can be described as follows in the context of graph partitioning problems. An initial solution is obtained, possibly at random. A series of N pairwise interchange of vertices is randomly made, and the gain is calculated after each interchange. The best solution is recorded among the N solutions generated, and the process is repeated. The procedure terminates when a prespecified termination criteria is satisfied. N Step Lookahead, like Simulated Annealing, has the advantage of being easily adapted to any combinatorial problem. However, only one parameter, namely N, has to be dealt with, as opposed to three parameters for Simulated Annealing.

N Step Lookahead and Simulated Annealing were implemented by Jackson and Kuh [12]. Different values of N were tested with $N = 7$ giving the best results. In the Simulated Annealing experiments, temperatures for the annealing schedule were obtained based on $T_{\text{new}} = \alpha T_{\text{old}}$, where α ranged between 0.9 and 0.99. The initial temperature T_0 was set, such that an initial exchange 20% worse was accepted with probability 0.9. The number of iterations per value of the temperature was empirically determined to be between 1 and 5. The method was halted when the cost was unchanged after 4 consecutive moves at a prespecified frozen state. In PARDIV, the parameter q was set equal to two.

The performance of the five procedures are compared in Table Ib. The times recorded are CPU seconds on an IBM 3090. The matching-based heuristics were coded in FORTRAN and compiled with the FORTVS compiler. The maximum weight matching routine used is an adaptation of the program SMP of Burkard and Derigs [3]. Both, N Step Lookahead and Simulated Annealing were coded in the C language. Note that the CPU times reported for N Step Lookahead and Simulated Annealing are when their best solutions were first obtained.

Table Ib shows that our matching-based heuristics are competitive with the random-based methods in terms of solution value. The average deviation ([average value of best known solution – average value of heuristic solution]/[average value of best known solution]) for each of our methods is approximately 2%. However, the matching-based heuristics dominate the probabilistic methods with respect to CPU time by factors of 10 to 3000! This is even more evident given the fact that the times reported for the probabilistic methods are the actual times at which the best solutions were found and not the total running times of the methods.

In these experiments, the method employing a matching heuristic (Heuristic 2) performs slightly better (solution value and time) than does the method using a matching algorithm (Heuristic 1). For all three of our techniques, the postprocessors significantly improved on the double-matching output. However, this does not imply that double matching is not essential. When random matching (as opposed to optimal or greedy matching) is performed and submitted to the postprocessors, the final solution values are observed to always be at least 15% below the best-known results. These observations are reinforced by our next set of experiments.

Dai and Kuh [6] have recently stated that their hierarchical method performs best in terms of speed and final design when each cluster contains four elements or less. We now turn our attention to this clustering problem. Five variations of each of the 10 industrial instances were generated. The variations were produced by randomly altering each nonzero edge weight. Note that the underlying graph of each of the instances is not changed. Table II presents these results. For each set of variations, the average final solution, the average percent improvement offered by the postprocessors, and the average percent deviation from the best-known solution are given. **Heuristic 4** starts with a random matching and then applies the postprocessors.

Heuristic 1, the method using a matching algorithm, consistently yields the best-known results. In comparison, Heuristic 4 consistently gives very poor solutions (24.5% below the best known). Note that this method relies entirely on the exchange-based postprocessors.

Table III describes five small randomly generated problems for which clusters of size four are desired. The problems vary from 12 to 20 nodes. For each problem, an edge was assigned a positive weight with probability varying between 0.1 and 0.4. The positive weights on the edges were drawn uniformly from the interval (1, 20). Table IV presents the corresponding results. The final solution value, the percent improvement offered by the postprocessors, and the CPU time in seconds on the IBM 3090 are reported. The optimal solutions (B&B) are also given. The optimal algorithm used to solve these problems involves a Lagrangian relaxation method imbedded within an implicit enumeration scheme [15]. The purpose of this algorithm is to provide a means of assessing the quality of the heuristic solutions obtained. Note that the size of the problems is limited to 20 nodes because of our

TABLE II. Aggregate results of five variations of each of the 10 industrial instances.

Instance		Heuristic 1	Heuristic 2	Heuristic 3	Heuristic 4
RAL 12	Avg. sol	4307	3927	4048	3172
	Imprv. %	6.2	32.6	25.5	44.3
	Error %	0.0	8.8	6.0	26.4
BARNS	Avg. sol.	3800	3640	3480	2980
	Imprv. %	10.0	23.1	32.2	57.0
	Error %	0.0	4.2	8.4	21.6
RAL 30	Avg. sol.	4166	3980	3243	3190
	Imprv. %	12.2	17.8	65.7	74.8
	Error %	0.0	4.5	22.2	23.4
RAL 33	Avg. sol.	5441	4974	4548	4321
	Imprv. %	6.0	12.4	30.9	88.0
	Error %	0.0	8.6	16.4	20.6
AMI 33	Avg. sol.	5295	4682	4535	4157
	Imprv. %	7.6	24.7	31.7	80.8
	Error %	0.0	11.6	14.3	21.5
AMI 49	Avg. sol.	10,681	9449	8515	8133
	Imprv. %	9.9	19.3	34.9	70.3
	Error %	0.0	11.5	20.3	23.9
RAL 49	Avg. sol.	2146	1933	1743	1520
	Imprv. %	7.5	16.7	37.8	87.0
	Error %	0.0	9.9	18.8	29.2
INTEL	Avg. sol.	6660	6213	4076	4603
	Imprv. %	6.4	15.8	95.7	93.9
	Error %	0.0	6.7	38.8	30.9
RAL 101	Avg. sol.	7686	7286	5696	6000
	Imprv. %	10.8	18.9	87.8	91.1
	Error %	0.0	5.2	25.9	21.9
RAL 122	Avg. sol.	8833	8326	7173	6590
	Imprv. %	10.6	16.4	47.4	92.8
	Error %	0.0	5.7	18.8	25.4
Overall	Imprv. %	8.7	19.8	49.0	78.0
	Error %	0.0	7.7	19.0	24.5

TABLE III. Graph characteristics of five small randomly generated problems.

Instance	No. nodes	Dummy nodes	Density (%)	Total edge weight
1	12	0	27	189
2	12	0	39	257
3	16	0	14	206
4	16	0	27	383
5	20	0	9	270

TABLE IV. Comparison between heuristics and branch and bound.

Instance		Heuristic 1	Heuristic 2	Heuristic 3	B&B
1	Value	113	113	113	113
	Imprv. %	7.0	16.8	14.1	—
	Sec	0.002	0.001	0.002	4.2
2	Value	139	139	139	139
	Imprv. %	0.0	25.0	2.1	—
	Sec	0.002	0.001	0.002	34
3	Value	164	151	164	164
	Imprv. %	0.0	0.0	0.0	—
	Sec	0.005	0.003	0.004	17
4	Value	207	205	194	207
	Imprv. %	5.7	11.2	5.6	—
	Sec	0.005	0.003	0.004	6000
5	Value	181	181	181	181
	Imprv. %	0.0	0.0	0.0	—
	Sec	0.009	0.005	0.009	850

inability to solve optimally larger problems in a reasonable amount of CPU time. The most prominent feature of Table IV is the high quality of the solutions produced by the double-matching heuristics. All problems are solved to optimality by Heuristic 1, with only two problems requiring help from the postprocessors.

5. EXTENSIONS AND CONCLUSION

In this article, we focused on the problem of partitioning the nodes of a graph into many equal-sized clusters so that the total weight of the edges running between clusters is minimized. The examples presented in the Introduction illustrate the practical importance of the problem, whereas the literature review indicates that little research on the topic has been conducted. Efficient transformations of this problem are developed and the complexity of the problem is proven. The experiments conducted with the tailored branch-and-bound algorithm show that the problem resists exact solution and should be approached heuristically. A class of matching-based, bounded approximation heuristics is proposed. The empirical performance of these heuristics is compared to probabilistic exchange procedures on both industrial and randomly generated instances. The results indicate that our heuristics perform extremely well, especially when CPU time is of concern. Their worst-case bound gives an intuitive justification for this performance.

Goldschmidt et al. have recently improved the bound on the optimal cluster weight (total edge weight in the clusters) from three to two for 3- and 4-dimensional matching [11]. Similarly, we have developed an $O(|V|^3)$ time, 2-approximate algorithm for $|V|/2$-dimensional matching (i.e., 2-partition). We also conjecture that the problem of finding an approximate solution within *100c* percent of the optimal partition weight is NP-hard for any constant c. If true, it implies that it will be difficult to find efficiently approximate feasible solutions to problems for which zero or near-zero weight cuts are important to the application. However, if the instances investigated have optimal solutions in which the cut weight is a significant proportion of the total edge weight in the graph, or the total edge weight in the clusters is of more interest to the application than is the actual weight of the cut, then the double-matching heuristics will be useful.

References

[1] D. Avis, Two greedy heuristics for the weighted matching problem. *Congr. Numer.* **XXI** (1978) 65–76.

[2] E. R. Barnes, An Algorithm for partitioning the nodes of a group. *SIAM J. Alg. Disc. Meth.* **3**(4) (1982) 541–550.

[3] R. E. Burkard and U. Derigs, *Assignment and Matching Problems: Solution Methods with Fortran-Programs,* Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin (1980).

[4] T. C. Chang and R. A. Wysk, *An Introduction to Automated Process Planning Systems.* Prentice-Hall, Englewood Cliffs, NJ (1985).

[5] C. C. Chen, Placement and partitioning methods for integrated circuit layout. Ph.D. Dissertation, Dept. of EECS, University of California, Berkeley, CA (1986).

[6] W. Dai, and E. Kuh, Simultaneous floor planning and global routing for hierarchical building block layout. *IEEE Trans. Comput. Aided Design ICs Syst.* (1987).

[7] T. A. Feo and M. Khellaf, The complexity of graph partitioning. Manuscript, Operations Research Group, Department of Mechanical Engineering, University of Texas at Austin (1988).

[8] L. R. Ford and D. R. Fulkerson, Maximal flow through a network. Rand Research Memorandum, RM-1400 (Nov., 1954).

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman & Company, New York (1979).

[10] M. R. Garey, D. S. Johnson, and L. S. Stokmeyer, Some simplified NP-complete graph problems. *Theor. Comput. Sci.* **1**(1) (1976) 237–267.

[11] O. Goldschmidt, M. Khellaf, and T. A. Feo, Private communication (1988).

[12] M. Jackson, and E. Kuh, A simple adaptive heuristic applied to K-way partitioning and building block placement. Manuscript, Computer Science Dept., University of California, Berkeley (1986).

[13] D. S. Johnson, C. H. Papadimitriou, P. Seymour, and M. Yannakakis, The complexity of multiway cuts. Manuscript, AT&T Bell Laboratories, Murray Hill, NJ (1984).

[14] B. W. Kernigham and S. Lin, An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **49**(2) (1970) 291–307.

[15] M. Khellaf, On the partitioning of graphs and hypergraphs. Ph.D. Dissertation, Dept. IEOR, University of California, Berkeley (1987).

[16] J. R. King and V. Nakornchai, Machine-component group formation in group technology: Review and extension. *Int. J. Prod. Res.* **20**(2) (1982) 117–123.

[17] M. Kirkpatrick, C. Gelatt, and M. Vecchi, Optimization by simulated annealing. *Science* **220**(4598) (1983).

[18] J. Kral, To the problem of segmentation of a program. *Info. Process. Machines* (1965).

[19] K. R. Kumar, A. Kusiak, and A. Vannelli, Grouping parts and components in flexible manufacturing system. *Eur. J. Operations Res.* **24** (1986) 387–397.

[20] R. M. MacGregor, On partitioning a graph: A heuristical and empirical study. Memorandum No. UCB/ERL M78/14, Electronics Research Laboratory, University of California, Berkeley (1978).

[21] M. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller, Equation of state calculation by fast computing machines. *J. Chem. Phys.* **21** (1953).
[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ (1982).
[23] R. E. Tarjan, *Data Structures and Network Algorithms.* CNMS-NSF Regional Conference Series in Applied Math. (1983).

Received November 1988
Accepted June 1989