A Primal-Dual Parallel Approximation Technique Applied to Weighted Set and Vertex Cover

Samir Khuller [*]   *Uzi Vishkin* [†]   *Neal Young* [‡]

Abstract

We give an efficient deterministic parallel approximation algorithm for the minimum-weight vertex- and set-cover problems and their duals (edge/element packing). The algorithm is simple and suitable for distributed implementation. It fits no existing paradigm for fast, efficient parallel algorithms — it uses only "local" information at each step, yet is *deterministic*. (Generally, such algorithms have required randomization.) The result demonstrates that linear-programming primal-dual approximation techniques can lead to fast, efficient parallel algorithms. The presentation does not assume knowledge of such techniques.

Keywords: set cover, vertex cover, parallel algorithms, approximation algorithms.

1 Introduction

The linear-programming primal-dual method for obtaining sequential algorithms for exact optimization problems is well studied [Ch83]. Primal-dual techniques have also been used to obtain sequential approximation algorithms (e.g., for NP-hard problems [Ch79, Ho82, etc.] and for on-line problems [You]). In this paper, we apply primal-dual techniques to obtain a deterministic parallel approximation algorithm for the minimum-weight vertex- and set-cover problems and their duals, maximum-weight edge and element packing.

The result demonstrates that linear-programming primal-dual techniques can lead to fast, efficient parallel algorithms. The algorithm is natural, yet fits no existing paradigm for such algorithms, being unique in that it uses only "local" information at each step, yet is

[*]Computer Science Department and Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, MD 20742. Part of the work of this author was done while this author was with UMIACS and partially supported by NSF grants CCR-8906949, CCR-9111348, and CCR-9103135. Email: samir@cs.umd.edu.

[†]Department of Electrical Engineering and Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, MD 20742. Also with Tel-Aviv University. Partially supported by NSF grants CCR-8906949 and CCR-9111348. Email: vishkin@umiacs.umd.edu.

[‡]Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, MD 20742. Partially supported by NSF grants CCR-8906949 and CCR-9111348. Email: young@umiacs.umd.edu.

deterministic. Generally, such algorithms have required randomization (e.g., [II86, ABI86, Lu86]).

Given an n-vertex, m-edge graph $G = (V, E)$ with vertex weights and an $\epsilon > 0$, our algorithm returns a vertex cover of weight at most $2/(1-\epsilon)$ times the minimum. It uses $O(\ln^2 m \ln \frac{1}{\epsilon})$ time and $m/\ln^2 m$ processors (i.e., $O(m \ln \frac{1}{\epsilon})$ operations) on an EREW-PRAM. More generally, the algorithm finds a set cover of weight at most $r/(1-\epsilon)$ times the minimum, using $O(r \ln^2 m \ln \frac{1}{\epsilon})$ time and $M/\ln^2 m$ processors (i.e., $O(r\, M \ln \frac{1}{\epsilon})$ operations). Here m is the number of elements, M is the sum of the set sizes, and r is the maximum number of sets in which any element occurs. (For vertex cover, $r = 2$.) In each case, the algorithm also implicitly finds a near-maximal dual solution (an edge or element *packing*) that is also within the corresponding factor of optimal.

The algorithm can be implemented using only integer arithmetic (see §4.2). If the weights are integers and $1/\epsilon$ is less than the sum of the vertex (resp. set) weights, then the weight of the cover is at most 2 (resp. r) times the minimum.

1.1 Related Work

The first r-approximation algorithm for weighted vertex/set cover was due to Hochbaum [Ho82]. She considered the relaxation of the natural integer linear program for the problem. The dual of this program is maximum edge packing. The so-called complimentary-slackness conditions are that a (fractional) cover and a packing are optimal provided (i) every vertex in the cover has its constraint met in the packing and (ii) every edge with non-zero packing weight has exactly one vertex in the cover. Hochbaum observed that an optimal packing was necessarily maximal, that for any maximal packing, the vertex set formed by the vertices whose packing constraints are met with equality form a cover, that such a packing and cover satisfy (i), and that (i) is sufficient to guarantee r-approximation because (ii) is approximately satisfied in that every edge has at most r vertices in the cover. Since an optimal dual solution can be found in polynomial time by solving the linear program, Hochbaum obtained a polynomial-time algorithm. Bar-Yehuda and Even [BE81] observed that sequentially raising the edge-packing weights as much as possible yields a maximal edge packing, thus obtaining a linear-time algorithm. For our algorithm, we relax (i) further, insisting only that every vertex in the cover *nearly* have its constraint met, and we show how to simultaneously raise *many* edge-packing weights so that the packing quickly becomes *nearly* maximal and the weight of the cover formed by the vertices that nearly have their packing constraints met with equality is within $r/(1-\epsilon)$ of optimal.

In [Cl83], Clarkson showed that in a restricted class of graphs, approximation ratios better than 2 could be obtained for vertex cover. Clarkson gave the first parallel approximation algorithm — a relatively complicated randomized algorithm [Cl91]. According to Motwani's lecture notes on approximation algorithms [Mot92], which contain a survey of results on vertex cover, the best approximation ratio known is $2 - \frac{\log\log n}{2\log n}$, due to Bar-Yehuda and Even [BE85] and to Monien and Speckenmeyer [MS85]. In [Ho83], Hochbaum gives a $(2-2/k)$-approximation algorithm, where k is the maximum vertex degree, and she conjectures that there is no polynomial-time c-approximation algorithm for any $c < 2$ unless P=NP.

Chvátal's weighted-set-cover algorithm guarantees a set cover of weight at most $\ln \Delta$ times the minimum, where Δ is the maximum set size [Ch79, Lo75, Jo74]. Berger, Rompel, and Shor [BRS89] give a parallel algorithm that guarantees a factor of $(1 + \epsilon) \ln \Delta$. Their algorithm uses a linear number of processors and runs in polylogarithmic time with some restrictions on the weights.

The intuition behind our complexity analysis relies on a lemma of general interest for parallel graph algorithms (Lemma 6). The lemma has previously found application in the analyses of randomized parallel graph algorithms: Israeli and Itai's maximum-matching algorithm [II86] and Alon, Babai and Itai's maximal-independent-set algorithm [ABI86].

In concurrent independent work, Cohen gives a parallel approximation algorithm for maximum flow in shallow networks [Co92]. If network flows are viewed as packings of source-to-sink paths, then maximal packings correspond to blocking flows. Cohen gives an ϵ-blocking flow algorithm that is similar in spirit to our algorithm, although a number of different issues arise. In a more recent work, Luby and Nisan give a parallel primal-dual approximation algorithm for positive linear programming [LN93]. Hochbaum's original algorithm can be parallelized by employing Luby and Nisan's algorithm; the resulting algorithm would obtain an approximation ratio comparable to ours and have an incomparable running time (growing linearly with $1/\epsilon$, but not with r). Previously, Goldberg *et al.* [GPST92] gave a parallel primal-dual algorithm to find (exactly) maximum-weight bipartite matchings. Their algorithm appears to be the first parallel algorithm to use primal-dual techniques, but it requires polynomial time.

1.2 Problem Definitions

Let $G = (V, E \subseteq 2^V)$ be a given hypergraph with vertex weights $w : V \rightarrow \Re^+$. Let $E(v)$ denote the set of edges incident to vertex v. Let G have m edges. Let r, the *rank* of G, be the maximum size of any edge. (For an ordinary graph, $r = 2$.) Let M, the *size* of G, be the sum of the edge sizes. For any real-valued function f and a subset S of its domain, let $f(S)$ denote $\sum_{x \in S} f(x)$.

Vertex Cover. A *vertex cover* for G is a subset $C \subseteq V$ of the vertices such that for each edge $e \in E$, some vertex in e is in C. The *(minimum-weight) vertex-cover problem* is to find a vertex cover with minimum total weight $w(C)$.

Edge Packing. An *edge packing* is an assignment $p : E \rightarrow \Re^+$ of non-negative weights to the edges of the hypergraph such that the total weight $p(E(v))$ assigned to the edges incident to any vertex v is at most $w(v)$. The *(maximum-weight) edge-packing problem* is to find an edge packing maximizing $p(E)$, the *weight* of p. The fractional relaxations of the vertex cover and edge packing problems are linear programming duals.

1.3 Related Problems

Let $\mathcal{C}$ be a family of sets with weights $w : \mathcal{C} \rightarrow \Re^{+}$. Let U denote $\bigcup_{S \in \mathcal{C}} S$.

Set Cover. A *set cover* is a subfamily $\mathcal{C}' \subseteq \mathcal{C}$ such that $\bigcup_{S \in \mathcal{C}'} S = U$ — in words, every element of U is in some set in the cover. The *(minimum-weight) set-cover problem* is to find a set cover of minimum total weight $w(\mathcal{C}')$.

Element Packing. An *element packing* is an assignment of non-negative weights to the elements such that the total weight assigned to the elements of any set S is at most $w(S)$. The *(maximum-weight) element-packing problem* is to find an element packing maximizing the net weight assigned to elements.

1.4 Equivalences

The vertex cover problem in hypergraphs is equivalent to the minimum-weight set-cover problem as follows. For each $S \in \mathcal{C}$ we have a vertex v_S in the hypergraph. For each element $x \in U$, we have an edge that *contains* v_S if and only if $x \in S$. The number of edges m is the number of elements. The rank r is the maximum number of sets in which any element occurs. The size M is the sum of the set sizes. The dual problems are also equivalent.

2 Reduction of Vertex Cover to ϵ-Maximal Packing

We first reduce our problem to the problem of finding what we call an ϵ-*maximal* packing. This reduction generalizes [Ho82, BE81], who considered $\epsilon = 0$.

Lemma 1 (Duality) *Let C be an arbitrary vertex cover and p an arbitrary edge packing. Then $p(E) \leq w(C)$.*

Proof:

$$p(E) = \sum_{e \in E} p(e) \leq \sum_{e \in E} |e \cap C|\, p(e) = \sum_{v \in C} p(E(v)) \leq \sum_{v \in C} w(v) = w(C).$$

□

Lemma 2 (Approximate Complimentary Slackness) *Let C be a vertex cover and p be a packing such that $p(E(v)) \geq (1-\epsilon) w(v)$ for every $v \in C$. Then $(1-\epsilon) w(C) \leq r\, p(E)$. By duality, the weights of C and p are within a factor of $r/(1-\epsilon)$ from their respective optima.*

Proof: Since $(1-\epsilon) w(v) \leq p(E(v))$ for $v \in C$,

$$(1-\epsilon) w(C) = (1-\epsilon) \sum_{v \in C} w(v) \leq \sum_{v \in C} p(E(v)) = \sum_{e \in E} |e \cap C|\, p(e) \leq r\, p(E).$$

□

We tighten Lemma 2 slightly when the weights are integers.

Lemma 3 *In Lemma 2, if the weights are integers and* $\epsilon < 1/w(V)$*, then the weight of* C *is at most* r *times the minimum.*

Proof: Let C^* be a minimum-weight cover. From Lemma 2, $(1-\epsilon)w(C) \leq r\,w(C^*)$, so $w(C) \leq \lfloor r\,w(C^*) + \epsilon\,w(C) \rfloor = r\,w(C^*)$. $\square$

Given a packing p, define $C_p = \{v \in V : p(E(v)) \geq (1-\epsilon)w(v)\}$. If C_p is a vertex cover, then we say p is *ϵ-maximal.* Note that p is 0-maximal if and only if p is maximal. By Lemma 2, if p is ϵ-maximal, then C_p and p are within a factor of $r/(1-\epsilon)$ from their respective optima.

3 The Algorithm

We have reduced the problem to finding an ϵ-maximal packing. The algorithm maintains a packing p and the partial cover $C_p = \{v \in V : p(E(v)) \geq (1-\epsilon)w(v)\}$. The algorithm increases the individual $p(e)$'s until p is ϵ-maximal and C_p is a cover. When a vertex v enters C_p, v and the edges containing v are deleted from the hypergraph. Let E_p denote the set of remaining edges, let $E_p(v)$ denote the remaining edges incident to vertex v, and let $d_p(v)$ be the degree of v in $G_p = (V, E_p)$. Define the *residual weight* $w_p(v)$ of vertex v to be $w(v) - p(E(v))$.

In a single round of the algorithm, for each remaining edge e, $p(e)$ is raised. To ensure that p remains a packing, each vertex v limits the increase in each $p(e)$ for $e \ni v$ to at most $w_p(v)/d_p(v)$. Each $p(e)$ is then increased as much as possible subject to the limits imposed by all the $v \in e$. That is, each $p(e)$ is increased by $\min_{v\in e} w_p(v)/d_p(v)$. The algorithm repeats this basic round until p converges to an ϵ-maximal packing. It then returns C_p. To implement the algorithm we maintain w_p instead of p:

Cover($G = (E, V), w, \epsilon$) — *Returns a vertex cover of hypergraph G of weight at most $r/(1-\epsilon)$ times the minimum.*

for $v \in V$ **par-do** $w_p(v) \leftarrow w(v)$; $E_p(v) \leftarrow E(v)$; $d_p(v) \leftarrow |E(v)|$
while edges remain **do**
  for each remaining edge e **par-do** $\delta(e) \leftarrow \min_{v\in e} w_p(v)/d_p(v)$
  for each remaining vertex v **par-do**
    $w_p(v) \leftarrow w_p(v) - \sum_{e\in E_p(v)} \delta(e)$
    if $w_p(v) \leq \epsilon\, w(v)$ **then**
      delete v and incident edges, updating $E_p(\cdot)$ and $d_p(\cdot)$
return the set of deleted vertices

As noted above, the limit on the increase in each $p(e)$ ensures that p remains a packing. Consequently, the correctness and the approximation ratio of the algorithm are established by Lemmas 2 and 3. Using standard techniques [Ja92], each iteration of the **while** loop beginning with q remaining edges can be done in $O(\ln q)$ time and $O(r\,q)$ operations on an EREW-PRAM.

4 Complexity Analysis

In this section, we prove our main theorem:

Main Theorem *The algorithm requires $O(r \ln^2 m \ln \frac{1}{\epsilon})$ time and $M/\ln^2 m$ processors, i.e., $O(r\, M \ln \frac{1}{\epsilon})$ operations.*

We use a potential function argument. Given a packing p, define

$$\phi_p = \sum_{v \in V} d_p(v) \ln \frac{w_p(v)}{\epsilon\, w(v)}.$$

The next lemma shows that during an iteration of the **while** loop ϕ_p decreases by at least the number of edges remaining at the end of the loop. This is how we show progress.

Lemma 4 *Let p and p', respectively, be the packing before and after an iteration of the* **while** *loop. Then $\phi_p - \phi_{p'} \geq |E_{p'}|$.*

Proof: During the iteration, we say that a vertex v *limits* an incident edge $e \in E_p$ if v determines the minimum in the computation of $\min_{v \in e} w_p(v)/d_p(v)$. For each vertex v, let v limit $L(v)$ edges, so that $w_{p'}(v) \leq w_p(v)(1 - L(v)/d_p(v))$. Let V' denote the set of vertices that remain after the iteration. Then

$$\begin{aligned}
\phi_p - \phi_{p'} &= \sum_{v \in V} \left(d_p(v) \ln \frac{w_p(v)}{\epsilon\, w(v)} - d_{p'}(v) \ln \frac{w_{p'}(v)}{\epsilon\, w(v)} \right) \\
&\geq \sum_{v \in V'} d_p(v) \ln \frac{w_p(v)}{w_{p'}(v)} \\
&\geq \sum_{v \in V'} -d_p(v) \ln(1 - L(v)/d_p(v)) \\
&\geq \sum_{v \in V'} L(v) \\
&\geq |E_{p'}|
\end{aligned}$$

The second-to-last step follows because $-\ln(1-x) \geq x$. The last step follows because each of the edges that remains is limited by some vertex in V'. □

Lemma 5 *There are at most $(1 + r \ln \frac{1}{\epsilon})(1 + \ln m)$ iterations.*

Proof: Let p and p', respectively, be the packing before and after any iteration. Let $a = r \ln \frac{1}{\epsilon}$.

Clearly $\phi_{p'} \leq |E_{p'}|\, a$. By Lemma 4, $\phi_{p'} \leq \phi_p - |E_{p'}|$. Thus, $\phi_{p'} \leq \phi_p(1 - 1/(a+1))$. Before the first iteration, $\phi_p \leq m\, a$. Inductively, before the ith iteration,

$$\phi_p \leq m\, a (1 - 1/(a+1))^{i-1} \leq m\, a \exp(-(i-1)/(a+1)).$$

The last inequality follows from $e^x \geq 1 + x$ for all x. Fixing $i = 1 + \lceil (a+1)\ln m \rceil$, we have $\exp(-(i-1)/(a+1)) \leq \exp(-\ln m) = 1/m$, so before the ith iteration, $\phi_p \leq a$.

During each subsequent iteration, at least one edge remains, so ϕ_p decreases by at least 1. Thus, $\phi_p \leq 0$ before an $i + a$th iteration can occur. $\square$

Time. As each iteration requires $O(\ln m)$ time, the above lemma implies that the total time is $O(r \ln^2 m \ln \frac{1}{\epsilon})$.

Operations. Recall that an iteration with q edges requires $O(r\,q)$ operations. Consequently, the total number of operations is bounded by an amount proportional to r times the sum, over all iterations, of the number of edges at the beginning of that iteration.

By Lemma 4, in a given iteration, ϕ_p decreases by at least the number of edges remaining at the end of the iteration. Thus, the sum over all iterations of the number of edges during the iteration is at most $m + \phi_p$ for the initial p. This is $m + M \ln \frac{1}{\epsilon}$. Hence there are $O(r\,M\,\ln \frac{1}{\epsilon})$ operations.

Processors. Using standard techniques, the operations can be efficiently scheduled without increasing the time or the operations by more than a constant, so by the Work-Time Scheduling Principle [Ja92], the number of processors required is $M/\ln^2 m$ — the work divided by the time. This establishes the Main Theorem.

4.1 The Intuition for Ordinary Graphs

The potential function analysis, while easy to verify, hides an interesting combinatorial principle that gives a good intuitive understanding of the algorithm for ordinary graphs. Recall that, during a single iteration of the **while** loop, a vertex v *limits* an edge e if v determines the minimum in the calculation of $\min_{v \in e} w_p(v)/d_p(v)$, and that $L(v)$ denotes the number of edges limited by v. If a vertex v limits at least a third of its incident edges, then $w_p(v)$ decreases by at least one third its value. Call such a vertex *good*. (After $O(\ln \frac{1}{\epsilon})$ iterations of the **while** loop in which v is good, v will enter C_p.) In a given iteration, few vertices might be good. However, at least half of the remaining edges touch good vertices. This is a consequence of the following lemma:

Lemma 6 ([II86, ABI86]) *Consider a directed graph. Call a vertex* good *if more than one-third of its incident edges are directed into it. Then at least half of the edges are directed into good vertices.*

Proof: If a vertex is not good, call it *bad*. The in-degree of any bad vertex is at most half its out-degree, so the number of edges directed into bad vertices is at most half the number of edges directed out of bad vertices. Thus, the number of edges directed into bad vertices is at most half the number of edges. Thus, at least half the edges are directed into good vertices. $\square$

To see why the lemma applies, imagine directing each remaining edge into a vertex that limits it. Then the lemma shows that at least half of the remaining edges touch vertices that are good. Thus, in a given iteration, at least half the edges touch vertices whose residual weights decrease by more than a factor of 1/3. This, intuitively, is why the algorithm makes progress.

This lemma is of independent interest: it drives the analyses of the running times of Israeli and Itai's randomized maximal matching algorithm [II86] and of Alon, Babai, and Itai's randomized maximal independent set algorithm [ABI86]. Interestingly, the natural generalization of the lemma to hypergraphs is not strong enough to give an analysis as tight as our potential function analysis.

4.2 Using Integer Arithmetic

If arithmetic precision is an issue, we can uniformly scale the original (integer) vertex weights so that the smallest weight is at least m/ϵ, and then use integer division (taking the floor) when computing the $w_p(v)/d_p(v)$'s. Essentially the same analysis carries through. (If $w(v) \geq m/\epsilon$, then $w_p(v) \geq m$ while v remains, so $w_p(v)/d_p(v) \geq 1$, hence $\lfloor w_p(v)/d_p(v) \rfloor \geq (w_p(v)/d_p(v))/2$, and the net reduction in a $w_p(v)$ during an iteration is at least half what it would have been without taking the floor. Thus, the analysis will go through by doubling the potential function.)

Assuming without loss of generality that $\epsilon \geq 1/(2w(V))$, if the original weights are k-bit integers, then the largest weight after scaling is bounded by

$$m \left\lceil \frac{1}{\epsilon} \right\rceil 2^{k+1} \leq 2^{k+2} m\, w(V) \leq 2^{2k+3} m\, |V|.$$

Hence the scaled weights are $(2k + 3 + \log_2 m + \log_2 |V|)$-bit integers. Subsequently all operations involve only integer arithmetic on smaller, non-negative integers.

Acknowledgments: We would like to thank Michael Luby and an anonymous referee for pointing out connections to the randomized algorithms of [ABI86, Lu86] and to the work by Hochbaum [Ho82], respectively.

References

[ABI86] N. Alon, L. Babai and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[BRS89] B. Berger, J. Rompel, and P. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. In *Proc. 30th Annual Symp. on Foundations of Computer Science*, pages 54–59, October 1989. Research Triangle Park, NC. To appear in *Journal of Algorithms* (Special issue on FOCS-89).

[BE81] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[BE85] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Math.*, 25:27–45, 1985.

[Ch79] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233-235, 1979.

[Ch83] V. Chvátal. *Linear Programming.* W. H. Freeman, 1983.

[Cl83] K. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16:23–25, 1983.

[Cl91] K. Clarkson. A randomized parallel algorithm for weighted vertex cover. Unpublished manuscript, October 1991.

[Co92] E. Cohen. Approximate max flow on small depth networks. In *Proc. 33rd Annual Symp. on Foundations of Computer Science*, pages 648–658, October 1992. Pittsburgh, PA.

[GPST92] A. Goldberg, S. Plotkin, D. Shmoys, É. Tardos. Interior point methods in parallel computation. *SIAM Journal on Computing*, 21(1):140–150, 1992.

[Ho82] D. Hochbaum. Approximation algorithms for set covering and vertex cover problems. *SIAM J. Computing*, 11:555-556, 1982.

[Ho83] D. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[II86] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, January 1986.

[Ja92] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[Jo74] D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Lo75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[Lu86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Computing*, 15(4):1036-1053, November 1986.

[LN93] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 448–457, May 1993. San Diego, CA.

[MS85] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.

[Mot92] R. Motwani. *Lecture notes on approximation algorithms.* Technical report #STAN-CS-92-1435. Dept. of Computer Science, Stanford University, 1992.

[You] N. Young. The k-server dual and loose competitiveness for paging. To appear in *Algorithmica*, (Special issue on on-line algorithms).