Distributed Approximation Algorithms via LP-duality and Randomization

Devdatt Dubhashi
Computer Science Department,
Chalmers University,
S-412 96 Göteborg, Sweden
E-mail: dubhashi@cs.chalmers.se

Fabrizio Grandoni and Alessandro Panconesi
Computer Science Department,
University of Rome "La Sapienza",
Via Salaria 113, 00198 ROME, Italy
E-mail: {grandoni, ale}@di.uniroma1.it

23rd March 2006

91.1 Introduction

The spread of computer networks, from sensor networks to the Internet, creates an ever growing need for efficient *distributed* algorithms. In such scenarios, familiar combinatorial structures such as spanning trees and dominating sets are often useful for a variety of tasks. Others, like maximal independent sets, turn out to be a very useful primitive for computing other structures. In a distributed setting, where transmission of messages can be orders of magnitude slower than local computation, the expensive resource is communication. Therefore the running time of an algorithm is given by the number of communication rounds that are needed by the algorithm. This will be made precise below.

In what follows we will survey a few problems and their solutions in a distributed setting: Dominating sets; edge and vertex colorings; matchings; vertex covers, and minimum spanning trees. These problems were chosen for a variety of reasons: They are fundamental combinatorial structures; computing them is useful in distributed settings; and they serve to illustrate some interesting techniques and methods.

Randomization, whose virtues are well-known to people coping with parallel and distributed algorithms, will be a recurrent theme. In fact, only rarely it has been possible to develop deterministic distributed algorithms for non-trivial combinatorial optimization problem. Here, in the section on vertex covers, we will discuss a novel and promising approach based on the primal-dual methodology to develop efficient, distributed deterministic algorithms. One of the main uses of randomization in distributed scenarios is to break the symmetry. This is well-illustrated in § 91.2. discussing dominating sets. Often the analysis of simple randomized protocols requires deep results from probability theory. This will be illustrated in § 91.3, where martingale methods are used to analyze some simple, and yet almost optimal, distributed algorithms for edge coloring. The area of distributed algorithms for graph problems is perhaps unique in complexity theory because it is possible to derive several non-trivial absolute lower-bounds (that is, not relying on special complexity assumptions such as $\mathsf{P} \neq \mathsf{NP}$). This will be discussed in § 91.6.

Let us then define the computation model. We have a message-passing, synchronous network: vertices are processors, edges are communication links and the network is synchronous. Communication proceeds in synchronous *rounds*: in each round, every vertex sends messages to its neighbors, receives messages from its neighbors, and does some amount of local computation. It is also assumed that each vertex has a unique identifier. In the case of randomized algorithms each node of the network has access to its own source of random bits. *In this model, the running time is the number of communication rounds.* This will be our notion of "time". As remarked, this is a very reasonable first approximation since typically sending a message is orders of magnitude slower than performing local computation.

Although we place no limits on the amount of local computation, the algorithms we describe perform polynomial-time local computations only. Under the assumption that local computations are polynomial-time several of the algorithms that we describe are "state-of-the-art", in the sense that their approximation guarantee is the same, or comparable, to that obtainable in a centralized setting. It is remarkable that this can be achieved in a distributed setting.

The model is in some sense orthogonal to the PRAM model for parallel computation where a set of polynomially-many, synchronous processors access a shared memory. There, communication is free: any two processors can communicate in constant time via the shared memory. In the distributed model, in contrast, messages are routed through the network and therefore the cost of sending a message is at least proportional to the length of the shortest path between the two nodes. On the other hand, local computation is inexpensive, while this is the expensive resource in the PRAM model.

Note that there is a trivial universal algorithm that always works: The network elects a leader which then collects the entire topology of the network, computes the answers, and notifies them to the other nodes. This will take a time proportional to the diameter of the network, which can be as large as n, the number of nodes. In general we will be looking for algorithms that take poly-logarithmically, in n, many communication rounds, regardless of the diameter of the network. Such algorithms will be called *efficient*.

Note the challenge here: if a protocol runs for t rounds then each processor can receive messages from nodes at distance at most t. For small values of t this means that the network is computing a global function of itself by relying on local information alone.

91.2 Small Dominating Sets

In this section we study the minimum dominating set problem. The advent of wireless networks gives a new significance to the problem since (connected) dominating sets are the structure of choice to set up the routing infrastructure of such ad-hoc networks, the so-called backbone (see, for instance, [1] and references therein). In the sequel we describe a nice algorithm from [2] for computing small dominating sets. The algorithm is in essence an elegant parallelization of the well-known greedy heuristic for set cover [3, 4]. Randomness is a key ingredient in the parallelization. The algorithm computes, on any input graph, a dominating set of size at most $O(\log \Delta)$*opt*, where as customary Δ denotes the maximum degree of the graph and *opt* is the smallest size of a dominating set in the input graph. By "computing a dominating set" we mean that at the end of the protocol every vertex decides whether it is in the dominating set or not. The algorithm was originally developed for the PRAM model but, as we will show, it can be implemented distributively. It is noteworthy that the approximation bound is essentially the "best possible" under the assumption that every node performs a polynomially-bounded computation during every round. "Best possible" means that an $o(\log n)$-approximation would imply that $\mathsf{P} = \mathsf{NP}$ [5], while a $(c \ln n)$-approximation, for a constant $c < 1$, would imply that NP could be solved exactly by means of slightly super-polynomial algorithms [6, 7].

We shall then describe a surprisingly simple deterministic algorithm that, building on top of the dominating set algorithm, computes a "best possible" connected dominating set, in $O(\log n)$ additional communication rounds [8].

There are other nice algorithms to compute dominating sets efficiently in a distributed setting. The algorithm in [9] is a somewhat different parallelization of the greedy algorithm, while [10] explores an interesting trade-off between the number of rounds of the algorithm and the quality of the approximation that it achieves. This paper makes use of LP-based methods, an issue that we will explore in Section 91.5.

91.2.1 Greedy

Let us start by reviewing the well-known greedy heuristic for set cover. Greedy repeatedly picks the set of minimum unit cost, creating a new instance after every choice by removing the points just covered. More formally, let (X, F, c) be a set cover instance where X is a ground set of elements and $F := \{S_i : \ S_i \subseteq X, i \in [m]\}$ is a family of non-empty subsets of X with positive costs $c(S) > 0$. The goal is to select a subfamily of minimum cost that covers the ground set. The cost of a subfamily is the sum of the costs of each set in the subfamily.

Dominating set is a special case of set cover. A graph G with positive weights $c(u)$, $u \in V(G)$, can be viewed as a set system $\{S_u : u \in V(G)\}$ with $S_u := N(u) \cup \{u\}$ where $N(u)$ is the set of neighbors of u, and $c(S_u) := c(u)$.

Given a set cover instance $I := (X, F, c)$, let $c(e) := \min_{e \in S \in F} \frac{c(S)}{|S|}$, be the *cost* of the element $e \in X$. This is the cheapest way to cover e where we do the accounting in the following natural way: when we pick a set, its cost is distributed equally to all elements it covers. An algorithm A may pick a certain set S' at this stage, then in this accounting scheme, each element $e \in S'$ pays the *price* $p(e) := \frac{c(S')}{|S'|}$. Once set s' is picked, we create a new instance I' with ground set $X' := X - \hat{S}$ and set system F' whose sets are defined as: $S'_i := S_i - \hat{S}$. The new costs coincide with the old ones: $c(S') = c(S)$, for all $S \in F'$. The algorithm continues in the same fashion until all elements are covered.

Greedy selects a set $\hat{S}$ at each stage that realizes the minimum unit cost i.e. $p(e) = c(e)$ at each stage. In other words, greedy repeatedly selects the set that guarantees the smallest unit price. For the discussion to follow concerning the distributed version of the algorithm it is important to notice that each element e is assigned a price tag $p(e)$ only once, at the time when it is covered by greedy. For a subset $A \subseteq X$, let $g(A) := \sum_{e \in A} p(e)$. Then $g(X)$, the sum of the unit prices, is the total cost incurred by greedy. The crux of the analysis is the next lemma.

Lemma 91.1 *For any set* S, $g(S) \leq H_{|S|} c(S)$ *where* $H_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$ *is the* k*-th harmonic number.*

Proof. Sort the elements of S according to the time when they are covered by greedy, breaking ties arbitrarily. Let $e_1, e_2, \ldots, e_k$ be this numbering. When greedy covers e_i it must be that $p(e_i) \leq \frac{c(S)}{k-i}$. The claim follows. □

Clearly we have that $g(A \cup B) \leq g(A) + g(B)$. Denoting with C^* an optimal cover, we have, by Lemma 91.1,

$$g(X) = g\left(\cup_{S \in C^*} S\right) \leq \sum_{S \in C^*} g(S) \leq \sum_{S \in C^*} H_{|S|} c(S) \leq \max_S H_{|S|} \sum_{S \in C^*} c(S) \leq \max_S H_{|S|} \mathit{opt}$$

It is well-known that $\log k \leq H_k \leq \log k + 1$. In the case of dominating set the bound becomes

$$g(X) \leq H_{\Delta+1}\mathit{opt} = O(\log \Delta)\mathit{opt}$$

where Δ is the maximum degree of the graph.

91.2.2 Greedy hordes

We now proceed to parallelize greedy. Figure 91.1 shows that the number of steps taken by greedy can be $\Omega(\sqrt{n})$. The problem lies in the fact that at any stage there is just one candidate set that gives the minimum unit cost $\hat{c}$. It is to get around this problem that we introduce the following notion. A *candidate* is any set S such that

$$\hat{c} \leq \frac{c(S)}{|S|} \leq 2\hat{c}. \tag{91.1}$$

Let us modify greedy in such a way that, at any step, it selects any set satisfying this condition. With this modification the solution computed by greedy will still be at most $O(\log n)\mathit{opt}$ since the algorithm pays at most twice the smallest unit price the overall we lose only a factor of two in the approximation.

Suppose now that the algorithm is modified in such a way that it adds to the solution all candidates satisfying (91.1). With this modification the graphs of Figure 91.1 will be covered in $O(\log n)$ steps. But as the example of the clique shows (all the nodes are selected) this increase in speed destroys the approximation guarantee. This is because the key requirement of the sequential greedy procedure is violated. In the sequential procedure the price $p(e)$ is paid only once, at the time when e is covered. If we do things in parallel we need to keep two conflicting requirements in mind: picking too many sets at once can destroy the approximation guarantee but picking too few can result in slow progress. And we must come up with a charging scheme to distribute the costs among the elements in a manner similar to the sequential case.

Rajagopalan and Vazirani solved this problem by devising a scheme that picks enough sets to make progress but at the same time retains the parsimonius accounting of costs like in the sequential version. Specifically, for every set S selected by greedy, the cost $c(S)$ will be distributed among the elements of a subset $T \subset S$ of at least $|S|/4$ elements. Crucially, the elements of T will be charged only once. If we can do this then we will lose another factor of four in the approximation guarantee with respect to greedy, all in all losing a factor of eight.

The scheme works as follows: line up the candidate sets satisfying (91.1) on one side and all the elements on the other. The elements are thought of as *voters* and cast their vote for one of the candidate sets containing them by an *election*. An election is conducted as follows:

- A random permutation of the candidates is computed.
- Among all the candidate sets that contain it, each voter votes for that set which has the lowest number in the permutation.
- A candidate is elected if it obtains at least $\frac{1}{4}$ of the votes of its electorate. Elected candidates enter the set cover being constructed.

The cost of the set can now be distributed equally among the elements that voted for it i.e at least a quarter of the elements.

Let us now describe the distributed implementation of this scheme in the specific case of the set system corresponding to the dominating set problem. During the execution nodes can be in four different states:

- They can be *free*. Initially all vertices are free.
- They can be *dominated*.
- They can be *dominators*. Dominators are added to the dominating set and removed from the graph.
- They can be *out*. Vertices are out when they are dominated and have no free neighbors. These vertices are removed from the graph since they can play no useful role.

The algorithm is a sequence of $\log \Delta$ *phases* during which the following invariant is maintained, with high probability. At the beginning of phase i, $i = 1, 2, \ldots, \log \Delta$, the maximum degree of the graph is at most $\Delta/2^{i-1}$. The candidates during phase i are all those vertices whose degree is in the interval $(\Delta/2^i, \Delta/2^{i-1}]$ i.e. they satisfy condition (91.1). Note that candidates can be free or dominated vertices. The voters are those free nodes that are adjacent to a candidate. This naturally defines a bipartite graph with candidates on one side, voters on the other, and edges that represent domination relationships. Each phase consists of a series of $O(\log n)$ elections. A free vertex can appear on both sides, since a free vertex can dominate itself. We shall refer to the neighbors of a candidate c in the bipartite graph as the *electorate* of c, and to the neighbors of a voter v as the *pool* of v. Election are carried out and elected candidates enter the dominating set.

Step 1 of each election seem to require global synchronization but a random permutation can be generated if the value of n is known. If each element picks a random number between 1 and n^k then with probability $1 - 1/n^{k-1}$ all choices will be distinct. Thus, the probability that there is a collision is negligible during the entire execution of the algorithm.

After every election nodes are removed for two different reasons. Elected nodes disappear from the candidate side of the bipartition, while their neighbors disappear from the other side, since they are no more free. In the analysis we will show that after one election the expected number of edges that disappear from the bipartite graph is a constant fraction of the total. This automatically implies that the total number of elections to remove all edges from the graph is $O(\log n)$ with overwhelming probability. More precisely, for any $c > 0$ there is $\alpha > 0$ such that, the probability that the bipartite graph is non-empty after $\alpha \log n$ elections is at most n^{-c} [11, 12]. It follows that α can be chosen in such a way that the probability that some phase does not end successfully is negligible.

A voter v is *influential* for a candidate c if at least $\frac{3}{4}$ of the voters in c's electorate have degree no greater than that of v. Let $d(v)$ denote the degree of v.

Lemma 91.2 *For any two voters v and w, $d(v) \geq d(w)$, in c's electorate,* $\mathsf{Pr}[w \text{ votes } c \mid v \text{ votes } c] \geq \frac{1}{2}$.

Proof. Let N_b denote the number of neighbors that v and w have in common, let N_v the number of neighbors of v that are not neighbors of w, and let N_w be the number of neighbors of w that are not neighbors of v. Then,

$$\mathsf{Pr}[w \text{ votes } c \mid v \text{ votes } c] = \frac{\mathsf{Pr}[w \text{ votes } c, v \text{ votes } c]}{\mathsf{Pr}[v \text{ votes } c]} = \frac{N_v + N_b}{N_v + N_b + N_w} \geq \frac{1}{2}.$$

□

Lemma 91.3 *Let v be an influential voter for c. Then,* $\mathsf{Pr}[c \text{ is elected} \mid v \text{ votes } c] \geq \frac{1}{6}$.

Proof. Let $X :=$ (# votes for c) and $Y := c - X$ where, with abuse of notation we use c to denote the size of c's electorate. Then, by Lemma 91.2

$$\mathsf{E}[X \mid v \text{ votes } c] \geq \sum_{w:\, d(w) \leq d(v)} \mathsf{Pr}[w \text{ votes } c \mid v \text{ votes } c] \geq \frac{3}{8}c.$$

Applying Markov's inequality to Y we get,

$$\begin{aligned} \mathsf{Pr}[c \text{ not elected} \mid v \text{ votes } c] &= \mathsf{Pr}[X < c/4 \mid v \text{ votes } c] = \mathsf{Pr}[Y \geq 3c/4 \mid v \text{ votes } c] \\ &\leq \frac{4\,\mathsf{E}[Y \mid v \text{ votes } c]}{3c} = \frac{4(c - \mathsf{E}[X \mid v \text{ votes } c])}{3c} \leq \frac{5}{6}. \end{aligned}$$

The claim follows. □

Lemma 91.4 *Fix a phase and let m denote the total number of edges in the bipartite graph at any stage in this phase. Let X denote the number of edges removed from the bipartite graph after one election. Then,* $\mathsf{E}[X] \geq \frac{m}{24}$.

Proof. An edge vc is *good* if v is influential for c. By definition, at least $\frac{1}{4}$ of the edges are good. Then,

$$\begin{aligned}
\mathsf{E}[X] &= \sum_{vc} \mathsf{Pr}[c \text{ is elected}, v \text{ votes } c] d(v) \\
&\geq \sum_{vc \text{ good}} \mathsf{Pr}[c \text{ is elected}, v \text{ votes } c] d(v) \\
&\geq \sum_{vc \text{ good}} \mathsf{Pr}[v \text{ votes } c]\ \mathsf{Pr}[c \text{ is elected} \mid v \text{ votes } c] d(v) \\
&= \sum_{vc \text{ good}} \mathsf{Pr}[c \text{ is elected} \mid v \text{ votes } c] \\
&\geq \frac{m}{24}, \quad \text{by Lemma 91.3.}
\end{aligned}$$

□

As remarked, this lemma implies that, with high probability, $O(\log n)$ rounds are sufficient for every phase. The resulting running time is $O(\log n \log \Delta)$ communication rounds, while the approximation guarantee is $O(\log \Delta)$. Vertices must know n to compute a permutation and to run the correct number of elections, and they must know Δ in order to decide whether they are candidates at the current phase. Alternatively, if only the value of n is known, the algorithm can execute $O(\log n)$ phases, for a total of $O(\log^2 n)$ many rounds.

91.2.3 Small Connected Dominating Sets

In this section we develop an efficient distributed algorithm for computing "best possible" connected dominating sets. Again, by this we mean that the protocol computes a connected dominating set of size at most $O(\log \Delta)$ times the optimum. Nowadays connected dominating sets are quite relevant from the application point of view since they are the solution of choice for setting up the backbones of self-organizing networks such as ad hoc and sensor networks (see [1] and references therein). A backbone is a subnetwork that is in charge of administering the traffic inside a network.

What is remarkable from the algorithmic point of view is that connectivity is a strong global property, and yet we will be able to obtain it by means of a distributed algorithm that relies on local information alone. The overall strategy can be summarized as follows:

- Compute a small dominating set.
- Connect it up using a sparse spanning network.

We saw in the previous section how to take care of Step 1. To connect up a dominating set we can proceed as follows. Let D be the dominating set in the graph G created after Step 1. Consider an auxiliary graph H with vertex set D and where any two $u, v \in D$ that are at distance $1, 2$ or 3 in G are connected by an edge in H. It is easy to see that H is connected if G is (which we assume). Every edge in H corresponds to a path with $0, 1$ or 2 vertices in G. If we inserted all such vertices we would still have a dominating set, since adding vertices can only improve domination. The resulting set would however be too large in general, since H can have as many as $|D|^2$ edges, each contributing with 2 vertices. The best way to connect D up would be to compute a spanning tree T. If we could do this, adding to D all vertices lying on paths corresponding to the edges of T, we would obtain the desired approximation since $E(T) = |D| - 1$ and recalling that $|D|$ is a $O(\log \Delta)$-approximation. Therefore, denoting with D^* and C^* an optimal dominating and connected dominating set, respectively, we would have (with some abuse of notation) that $|D \cup V(T)| \le 3|D| \le O(\log \Delta)|D^*| \le O(\log \Delta)|C^*|$.

The problem however is that, as we discuss in Section 91.6.1, computing a spanning tree takes time $\Omega(\sqrt{n})$. In what follows we show a very simple algorithm that computes, in $O(\log |V(G)|)$ many communication rounds, a network $S \subset H$ such that (a) S is connected, (b) $|E(S)| = O(|D|)$ and (c) $V(S) = D$. In words, S is a sparse connected network that spans the whole of D with linearly many edges. If we can compute such an S than we will have a connected dominating set of size at most $O(\log \Delta)$ times the optimum. S will not be acyclic but this is actually a positive thing since it makes S more resilient to failures. In fault-prone environments such as ad hoc and sensor networks this kind of redundancy is actually very useful. The key to computing S is given by the following lemma (see, for instance, [13, Lemma 15.3.1]). Recall that the *girth* of a graph G is the length of the shortest cycle in G.

Lemma 91.5 *Let $G = (V, E)$ be a graph of girth g, and let $m := |E|$ and $n := |V|$. Then, $m \le n + n^{1+2/(g-1)}$.*

Proof. Assume $g = 2k + 1$ and let $d := \frac{m}{n}$. Consider the following procedure. As long as there is a vertex whose degree is less than d, remove it. Every time we remove a vertex the new minimum degree is at least as large as the old one. Therefore this procedure ends with a graph whose minimum degree is at least d. Pick now any vertex in this graph and start a breadth first search. This generates a tree in which the root has at least d children and every other node has at least $d-1$ children. Moreover, assigning level 0 to the root, this tree is a real tree up to and including level $k-1$, i.e. no two vertices of this BFS exploration coincide up to that level. Therefore

$$n \geq 1 + d + d(d-1) + \ldots + d(d-1)^{k-1} \geq (d-1)^k.$$

Recalling the definition of d, the claim follows. The proof for the case $g = 2k$ is analogous. □

Note that if $g = 2\log n + 1$ then $m \leq 3n$. Define a cycle to be *small* if it is of length at most $2\log n + 1$. The following amazingly simple protocol removes all small cycles while, crucially, preserving connectivity:

- If an edge is the smallest in some small cycle, it is deleted.

Assume that every edge in the graph has a unique identifier. An edge is smaller than an another edge if its identifier is smaller than that of the other edge. It is clear that every small cycle is destroyed. The next lemma shows that connectivity is preserved.

Lemma 91.6 *The above protocol preserves connectivity.*

Proof. Sort the edges by increasing ID's and consider the following sequential procedure. At the beginning all edges are present in the graph. At step i edge e_i is considered. If e_i is in a small cycle then it is removed. This breaks all small cycles and preserves connectivity, since an edge is removed only when there is another path connecting its endpoints. The claim follows by observing that the sequential procedure and the distributed protocol remove the same set of edges. □

To implement the protocol we only need to determine the small cycles to which an edge belong. This can be done by a BFS of depth $O(\log n)$ starting from every vertex. If edges do not have distinct ID's to start with they can be generated by selecting a random number in the range $[m^3]$ which ensures that with all ID's are distinct with overwhelming probability. This requires the value of n or m to be known. This sparsification technique appears to be quite effective in practice [1].

91.3 Coloring: The Extraordinary Career of a Trivial Algorithm

Consider the following sequential greedy algorithm to color the vertices of an input graph with $\Delta + 1$ colors, where Δ is the maximum degree: pick a vertex, give it a color not assigned to any of its neighbors; repeat until all vertices are colored. In general Δ can be quite far from the optimal value $\chi(G)$ but it should not be forgotten that the chromatic number is one of the most difficult combinatorial problems to approximate [14, 15, 16].

In this section we will see how efficient distributed implementations of this simple algorithm lead to surprisingly strong results for vertex and especially edge coloring. Consider first the following distributed implementation. Each vertex u is initially given a list of colors $L_u := \{1, 2, \ldots, \Delta + 1\}$. Computation proceeds in rounds, until the graph is colored. One round is as follows: each uncolored vertex u picks a tentative color $t_u \in L_u$; if no neighboring vertex has chosen the same tentative color, t_u becomes the final color of u, and u stops. Otherwise L_u is updated by removing from it all colors assigned to neighbors of u at the current round. We shall refer to this as the *trivial algorithm*. It is apparent that the algorithm is distributed.

The trivial algorithm is clearly correct. An elementary, but non-trivial analysis shows that the probability that an uncolored vertex colors itself in one round is at least $\frac{1}{4}$ [17]. As we discussed in the previous section, this implies that the algorithm will color the entire network within $O(\log n)$ communication rounds, with high probability.

The following slight generalization is easier to analyze. At the beginning of every round, uncolored vertices are *asleep* and wake up with probability p. The vertices that wake up execute the round exactly as described before. At the end of the round, uncolored vertices go back to sleep. Said it differently, the previous algorithm is obtained by setting $p = 1$. In the sequel we will refer to this generalization as the (generalized) trivial algorithm. Luby analyzed this algorithm for $p = \frac{1}{2}$ [18]. Heuristically it is not hard to see why the algorithm makes progress in this case. Assume u is awake. The expected number of neighbors of u that wake up is $d(u)/2 \leq |L_u|/2$.

In the worst case, these neighbors will pick different colors and all these colors will be in L_u. Even then, u will have probability at least $\frac{1}{2}$ to pick a color that creates no conflict. Thus, with probability $\frac{1}{2}$ a vertex wakes up and, given this, with probability at least $\frac{1}{2}$ it colors itself. The next proposition formalizes this heuristic argument.

Proposition 91.1 *When $p = \frac{1}{2}$ the probability that an uncolored vertex colors itself in one round is at least $\frac{1}{4}$.*

Proof. Let t_u denote the tentative color choice of a vertex u.

$$\begin{aligned}
\Pr[u \text{ does not color} \mid u \text{ wakes up}] &= \Pr[\exists v \in N(u)\ t_u = t_v \mid u \text{ wakes up}] \\
&\leq \sum_{v \in N(u)} \Pr[t_u = t_v \mid u \text{ wakes up}] \\
&= \sum_{v \in N(u)} \Pr[t_u = t_v \mid u \text{ and } v \text{ wake up}] \Pr[v \text{ wakes up}] \\
&= \sum_{v \in N(u)} \frac{|L_u \cap L_v|}{|L_v||L_u|}\frac{1}{2} \leq \sum_{v \in N(u)} \frac{1}{|L_u|}\frac{1}{2} \leq \frac{1}{2}.
\end{aligned}$$

Therefore,

$$\Pr[u \text{ colors itself}] = \Pr[u \text{ colors itself} \mid u \text{ wakes up}] \Pr[u \text{ wakes up}] \geq \frac{1}{4}.$$

□

Note that the trivial algorithm works just as well if the lists are initialized as $L_u := \{1, 2, \ldots, d(u) + 1\}$, for all $u \in V(G)$, for any value of $p > 0$. Interestingly, in practice with $p = 1$ the trivial algorithm is much faster than Luby's one. In fact, experimentally, the speed of the algorithm increases regularly and monotonically as p tends to 1 [19].

In the distributed model we can simulate the trivial algorithm for the line graph with constant-time overhead. In this case the algorithm will be executed by the edges rather than the vertices, each edge e having its own list L_e. In this fashion we can compute edge colorings that are approximated by a factor of 2 (since $2\Delta - 1$ colors are used). It is a challenging open problem whether an $O(\Delta)$-approximation can be computed deterministically in the distributed model. The best known result so far is an $O(\Delta \log n)$-approximation [20]. But the real surprise is that the trivial algorithm computes near-optimal edge colorings!

Vizing's Theorem shows that every graph G can be edge colored sequentially in polynomial time with Δ or $\Delta + 1$ colors (see, for instance, [21]). The proof is in fact a polynomial time sequential algorithm for achieving a $\Delta + 1$ coloring. Thus edge coloring can be well approximated. It is a very challenging open problem whether colorings as good as these can be computed fast in a distributed model.

If the edge lists L_e's are initialized to contain just a bit more than Δ colors, say $|L_e| = (1 + \epsilon)\Delta$ for all e, then the trivial algorithm will edge color the graph within $O(\log n)$ communication rounds. Here ϵ can be any fixed, positive constant. Some lists can run out of colors and, consequently, the algorithm can fail, but this happens with a probability that goes to zero as n, the number of vertices, grows. All this is true, provided that the minimum degree $\delta(G)$ is large enough, i.e. $\delta(G) \gg \log n$ [22, 23]. For Δ-regular graphs the condition becomes $\Delta \gg \log n$.

In fact, the trivial algorithm has in store more surprises. If the input graph is Δ-regular and has no triangles, it colors the vertices of the graph using only $O(\Delta/\log\Delta)$ colors. This is in general optimal, since there are infinite families of triangle-free graphs that need these many colors [24]. Again, the algorithm fails with negligible probability, provided that $\Delta \gg \log n$. For the algorithm to work, the value of p must be set to a value that depends on the round: small initially, it grows quickly to 1 [25].

The condition $\Delta \gg \log n$ appears repeatedly. The reason is that these algorithms are based on powerful martingale inequalities and this condition is needed to make them work. These probabilistic inequalities are the subject of the next section.

91.3.1 Coloring with Martingales

Let $f(X_1,\ldots,X_n)$ be a function for which we can compute $\mathsf{E}[f]$, and let the X_i's be independent. Assume moreover that the following Lipshitz condition (with respect to the Hamming distance) holds:

$$|f(X) - f(Y)| \le c_i \tag{91.2}$$

whenever $X := (x_1,\ldots,x_n)$ and $Y := (y_1,\ldots,y_n)$ differ only in the i-th coordinate. Then, f is sharply concentrated around its mean:

$$\Pr[|f - \mathsf{E}[f]| > t] \le 2e^{-2t^2/\sum_i c_i^2}. \tag{91.3}$$

This is the simplest of a series of powerful concentration inequalities dubbed the Method of Bounded Differences (MOBD) [26]. The method is based on martingale inequalities (we refer the reader to the thorough and quite accessible treatment in [12]). In words, if a function does not depend too much on any coordinate then it is almost constant.

To appreciate the power and ease of use of (91.3) we derive the well known Chernoff-Hoeffding bound (see, among others, [27, 28, 12]). This bound states that if $X := \sum_{i=1}^n X_i$ is the sum of independent, binary random variables $X_i \in \{0,1\}$, then X is concentrated around its mean: $\Pr[|X - \mathsf{E}[X]| > t] \le 2e^{-2t^2/n}$. This captures the well-known fact that if a fair coin is flipped many times we expect HEADS to occur roughly 50% of the time, and this bound gives precise probability estimates of deviating from the mean. This bound can be recovered from (91.3) simply by defining $f := X$ and by noticing that condition 91.2 holds with $c_i = 1$.

We now apply the MOBD to the analysis of the trivial algorithm in a simplified setting. Let us assume that the network is a triangle-free, d-regular graph. We analyze what happens to the degree of a node after the first round. The probability with which an edge colors itself is $\left(1-\frac{1}{d}\right)^{2d-2} \sim \frac{1}{e^2}$. Therefore, denoting with f the new degree of vertex u, we have that $\mathsf{E}[f\,] = \Theta(d)$. At first blush it may seem that the value of f depends on the tentative color choices of $\Theta(d^2)$ edges: those incident on u and the edges incident on them. But it is possible to express f as a function of $2d$ variables only, as follows. For every $v \in N(u)$ consider the bundle of $d-1$ edges incident on v that are not incident on u, and treat this bundle as a single random variable, denoted as B_v. B_v is a random vector with $d-1$ components, each specifying the tentative color choice of an edge incident on v (except uv). Furthermore, for every edge $e = uv$, let X_e denote e's color choice. Thus, f depends on d variables of type X_e and on d variables of type B_v. What is the effect of these variables on f? If we change the value of a fixed X_e, and keep all remaining variables the same, this color change can affect at most two edges (one of which is e itself). The resulting c_e is 2. The cumulative effect of the first d variables of type X_e is therefore $4d$.

Note now, that since the network is triangle-free, changing the value of a bundle B_v can only affect the edge uv the bundle is incident to. Thus the effect of changing B_v while keeping everything else fixed, is 1. Summing up we get a total effect of $\sum_i c_i^2 = 5d$. Plugging in this value in (91.3), for $t = \epsilon d$, where $1 > \epsilon > 0$ we get, $\Pr[|f - \mathsf{E}[f\,]| > \epsilon d] \leq 2e^{-2\epsilon^2 d/5}$. We can see here why it is important to have $d \gg \log n$. With this condition, the bound is strong enough to hold for all vertices and all rounds simultaneously. In fact a value $d = \Theta(\log n)$ would seem to be enough, but the error terms accumulate as the algorithm progresses. To counter this cumulative effect, we must have $d \gg \log n$.

This establishes that the graph stays almost regular after one round (and in fact at all times), with high probability. For the full analysis one has to keep track of several other quantities besides vertex degrees, such as the size of the color lists. While the full analysis of the algorithm is beyond the scope of this survey, this simple example already clarifies some of the issues. For instance, if the graph is not triangle-free, then the effect of a bundle can be much greater than 1. To cope with this, more powerful inequalities, and a more sophisticated analysis, are needed [22, 12, 29, 23]. We remark that in general these inequalities do not even require the variables X_i to be independent. In fact, only the following bounded difference condition is required,

$$|\mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i = a] - |\mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i = b]| \leq c_i.$$

If this condition holds for all possible choices of a and b, and for all i, then Equation 91.3 follows. What is behind this somewhat contrived definition is the fact that the sequence $Y_i := \mathsf{E}[f|X_1, \ldots, X_{i-1}, X_i]$ is a martingale (the so-called Doob martingale). A martingale is simply a sequence of random variables $Z_0, Z_1, \ldots, Z_n$ such that $\mathsf{E}[Z_i|Z_0, \ldots, Z_{i-1}] = Z_i$, for $i = 1, 2, \ldots, n$. A typical example of a martingale is a uniform random walk in the integer lattice, where a particle can move left, right, up or down with equal probability. If Z_i denotes the distance of the particle from the origin, the expected distance after one step stays put. A close relative of the Chernoff-Hoeffding bound, known as Azuma's Inequality, states that if a martingale sequence $Z_0, Z_1, \ldots, Z_n$ satisfies the bounded difference condition $|Z_i - Z_{i-1}| \leq c_i$ for $i = 1, 2, \ldots, n$, then it is unlikely that Z_n is far from Z_0:

$$\Pr[|Z_n - Z_0| > t] \leq 2e^{-2t^2/\sum_i c_i^2}. \tag{91.4}$$

In words, if a martingale sequence does not make big jumps, then it is unlikely to stray afar from its starting point. This is true for the random walk; it is very unlikely that after n steps the particle will be far from the origin. Note that for a Doob martingale $Y_0 = \mathsf{E}[f\,]$ and $Y_n = f$, so that Equation 91.4 becomes Equation 91.3.

To see the usefulness of this more awkward formulation, let us drop the assumption that the network is triangle-free and analyze again what happens to the vertex degrees, following the analysis from [29]. As observed this introduces the problem that the effect of bundles can be very large: changing the value of B_v can affect the new degree by as much as $d - 1$. We will therefore accept the fact that the new degree of a vertex is a function of $\Theta(d^2)$ variables, but we will be able to bound the effect of edges at distance one from the vertex. Fix a vertex v and let $N^1(v)$ denote the set of "direct" edges– i.e. the edges incident on v– and let $N^2(v)$ denote the set of "indirect edges" that is, the edges incident on a neighbor of v. Let $N^{1,2}(v) := N^1(v) \bigcup N^2(v)$. Finally, let $T := (T_{e_1}, \ldots, T_{e_m})$, $m = |E(G)|$, be the random vector specifying the tentative color choices of the edges in the graph G. With this notation, the number of edges successfully colored at vertex v is a function $f(T_e, e \in N^{1,2}(v))$ (to study f or the new degree is the same: if f is concentrated so is the new degree).

Let us number the variables so that the direct edges are numbered *after* the indirect edges (this will be important for the calculations to follow). We need to compute

$$\lambda_k := |\mathsf{E}[\mathsf{f} \mid \boldsymbol{T}_{\mathsf{k}-1}, \mathsf{T}_\mathsf{k} = \mathsf{c}_\mathsf{k}] - \mathsf{E}[\mathsf{f} \mid \boldsymbol{T}_{\mathsf{k}-1}, \mathsf{T}_\mathsf{k} = \mathsf{c}'_\mathsf{k}]|. \tag{91.5}$$

We decompose f as a sum to ease the computations later. Introduce the indicator functions $f_e, e \in E$: $f_e(\boldsymbol{c})$ is 1 if edge e is successfully colored in coloring $\boldsymbol{c}$, and 0 otherwise. Then $f = \sum_{v \in e} f_e$. Hence we are reduced, by linearity of expectation, to computing for each $e \in N^1(v)$, $|\mathsf{Pr}[f_e = 1 \mid \boldsymbol{T}_{k-1}, T_k = c_k] - \mathsf{Pr}[f_e = 1 \mid \boldsymbol{T}_{k-1}, T_k = c'_k]|$.

To compute a good bound for λ_k in (91.5), we shall lock together two distributions Y and Y'. Y is distributed as $\boldsymbol{T}$ conditioned on $\boldsymbol{T}_{k-1}, T_k = c_k$, and Y', while Y' distributed as $\boldsymbol{T}$ conditioned on $\boldsymbol{T}_{k-1}, T_k = c'_k$. We can think of $\boldsymbol{Y'}$ as identically equal to $\boldsymbol{Y}$ except that $\boldsymbol{Y'}_k = c'_k$. Such a pairing (Y, Y') is called a *coupling* of the two different distributions $[T|\boldsymbol{T}_{k-1}, T_k = c_k]$ and $[T|\boldsymbol{T}_{k-1}, T_k = c'_k]$. It is easily seen that by the independence of all tentative colors, the marginal distributions of $\boldsymbol{Y}$ and $\boldsymbol{Y'}$ are exactly the two conditioned distributions $[\boldsymbol{T} \mid \boldsymbol{T}_{k-1}, T_k = c_k]$ and $[\boldsymbol{T} \mid \boldsymbol{T}_{k-1}, T_k = c'_k]$ respectively. Now let us compute $|\mathsf{E}[\mathsf{f}(\boldsymbol{Y}) - \mathsf{f}(\boldsymbol{Y'})]|$.

First, let us consider the case when $e_1, \ldots, e_k \in N^2(v)$, i.e. only the choices of indirect edges are exposed. Let $e_k = (w, z)$, where w is a neighbor of v. Then for a direct edge $e \neq vw$, $f_e(\boldsymbol{y}) = f_e(\boldsymbol{y'})$ because in the joint distribution space, $\boldsymbol{y}$ and $\boldsymbol{y'}$ agree on all edges incident on e. So we only need to compute $|\mathsf{E}[\mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y}) - \mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y'})]|$. To bound this simply, we observe first that $f_{vw}(\boldsymbol{y}) - f_{vw}(\boldsymbol{y'}) \in [-1, 1]$ and second that $f_{vw}(\boldsymbol{y}) = f_{vw}(\boldsymbol{y'})$ unless $y_{vw} = c_k$ or $y_{vw} = c'_k$. Thus we can conclude that $\mathsf{E}[\mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y}) - \mathsf{f}_{\mathsf{vw}}(\boldsymbol{Y'})]| \leq \mathsf{Pr}[\mathsf{Y_e} = \mathsf{c_k} \vee \mathsf{Y_e} = \mathsf{c'_k}] \leq \frac{2}{\mathsf{d}}$.

In fact one can do a tighter analysis using the same observations. Let us denote $f_e(\boldsymbol{y}, y_{w,z} = c_1, y_e = c_2)$ by $f_e(c_1, c_2)$. Note that $f_{vw}(c_k, c_k) = 0$ and similarly $f_{vw}(c'_k, c'_k) = 0$. Hence

$$\begin{aligned}
\mathsf{E}[\mathsf{f_e}(\boldsymbol{Y}) - \mathsf{f_e}(\boldsymbol{Y'}) \mid \mathsf{z}] &= (f_{vw}(c_k, c_k) - f_{vw}(c'_k, c_k))\mathsf{Pr}[Y_e = c_k] + (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c'_k))\mathsf{Pr}[Y_e = c'_k] \\
&= (f_{vw}(c_k, c'_k) - f_{vw}(c'_k, c_k))\frac{1}{d}.
\end{aligned}$$

(Here we used the fact that the distribution of colors around v is unaffected by the conditioning around z and that each color is equally likely.) Hence $|\mathsf{E}[\mathsf{f_e}(\boldsymbol{Y}) - \mathsf{f_e}(\boldsymbol{Y'})]| \leq \frac{1}{\mathsf{d}}$.

Now let us consider the case when $e_k \in N^1(v)$, i.e. choices of all indirect edges and of some direct edges have been exposed. In this case, we merely observe that f is Lipshitz with constant 2: $|f(\boldsymbol{y}) - f(\boldsymbol{y'})| \leq 2$ whenever $\boldsymbol{y}$ and $\boldsymbol{y'}$ differ in only one coordinate. Hence we can easily conclude that $|\mathsf{E}[\mathsf{f}(\boldsymbol{Y}) - \mathsf{f}(\boldsymbol{Y'})]| \leq 2$.

Overall, $\lambda_k \leq 1/d$ for an edge $e_k \in N^2(v)$, and $\lambda_k \leq 2$ for an edge $e_k \in N^1(v)$. Therefore we get

$$\sum_k \lambda_k^2 = \sum_{e \in N^2(v)} \frac{1}{d^2} + \sum_{e \in N^1(v)} 4 \leq 4d + 1.$$

We thus arrive at the following sharp concentration result by plugging into Equation 91.3: Let v be an arbitrary vertex and let f be the number of edges successfully colored around v in one stage of the trivial algorithm. Then,

$$\mathsf{Pr}[|f - \mathsf{E}[\mathsf{f}]| > \mathsf{t}] \leq 2\exp\left(-\frac{\mathsf{t}^2}{2\mathsf{d} + \frac{1}{2}}\right).$$

Since $\mathsf{E}[f\,] = \Theta(d)$, this is a very strong bound.

91.4 Matchings

Maximum matching is probably one of the best studied problems in Computer Science: given a weighted undirected graph $G = (V, E)$, compute a subset of pairwise non-incident edges (*matching*) of maximum cost. For the sake of simplicity, we will focus on the the cardinality version of the problem, where all the edges have weight one.

It is not hard to show that a maximum matching cannot be computed efficiently (i.e. in poly-logarithmic time) in a distributed setting.

Lemma 91.7 *Any distributed maximum matching algorithm requires $\Omega(n)$ rounds.*

Proof. Consider the following *mailing problem*: let P be a path of $n = 2k + 1$ nodes, and let ℓ and r be the left and right endpoints of the path, respectively. Let moreover c be the *central* node of the path. Nodes ℓ and r receive the same input bit b, and the problem is to forward b to the central node c. Clearly this process takes at least k rounds.

Now assume by contradiction that there exists a $o(n)$ distributed maximum matching protocol $\mathcal{M}$. We can use $\mathcal{M}$ to solve the mailing problem above in the following way. All the nodes run $\mathcal{M}$ on the auxiliary graph $P(b)$ obtained from P by removing the edge incident to ℓ if $b = 1$, and the edge incident to r otherwise. If $b = 1$ ($b = 0$), the edge on the left (right) of v must belong to the (unique) maximum matching. This way c can derive the value of the input bit b in $o(n) = o(k)$ rounds, which is a contradiction. □

Fischer, Goldberg, Haglin, and Plotkin [30] described a parallel algorithm to compute a near-optimal matching in arbitrary graphs. Their algorithm can be easily turned into a distributed protocol to compute a $k/(k+1)$-approximate solution in poly-logarithmic time, for any fixed positive integer $k > 0$. A crucial step in the algorithm by Fischer et al. is computing (distributively) a maximal independent set. Since this sub-problem is rather interesting by itself in the distributed case, in Section 91.4.1 we will sketch how it can be solved efficiently. In Section 91.4.2 we will describe and analyze the algorithm by Fischer et al.

91.4.1 Distributed Maximal Independent Set

Recall that an *independent set* of a graph is a subset of pairwise non-adjacent nodes. No deterministic protocol is currently known for the problem. Indeed, this is one of the main open problems in distributed algorithms. Luby [31] and independently Alon, Babai, and Itai [32] gave the first distributed randomized algorithms to compute a maximal independent set. Here we will focus on Luby's result, as described in Kozen's book [33].

As the algorithm by Fischer et al., Luby's algorithm was originally thought for a parallel setting, but it can be easily turned into a distributed algorithm. It is worth to notice that transforming an efficient parallel algorithm into an efficient distributed algorithm is not always trivial. For example there is a deterministic parallel version of Luby's algorithm, while, as mentioned above, no efficient deterministic distributed algorithm is known for the maximal independent set problem.

Luby's algorithm works in stages. In each stage one (not necessarily maximal) independent set I is computed, and the nodes I are removed from the graph together with all their neighbors. All the edges incident to deleted nodes are also removed. The algorithm ends when no node is left. At the end of the algorithm a maximal independent set is given by the union of the independent sets I computed in the different stages.

It remains to describe how each independent set I is computed. Each node v in the (current) graph independently becomes a *candidate* with probability $\frac{1}{2d(v)}$. Then, for any two adjacent candidates, the one of lower degree is discarded from S (ties can be broken arbitrarily). The remaining candidates form the set I.

Each stage can be trivially implemented with a constant number of communication rounds. The expected number of rounds is $O(\log n)$. More precisely, in each stage at least a constant expected fraction of the (remaining) edges are removed from the graph.

A crucial idea in Luby's analysis is the notion of good nodes: a node v is *good* if at least one third of its neighbors have degree not larger than v. In particular, this implies

$$\sum_{u\in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6}. \tag{91.6}$$

Otherwise v is *bad*. Though there might be few good nodes in a given graph, the edges incident to at least one good node are a lot. Let us call an edge *good* if it is incident to at least one good node, and *bad* otherwise. The following lemma holds.

Lemma 91.8 *At least one half of the edges are good.*

Proof. Direct all the edges toward the endpoint of higher degree, breaking ties arbitrarily. Consider any bad edge e directed toward a given (bad) node v. By definition of bad nodes, the out-degree of v is at least twice its own in-degree. Thus we can uniquely map e into a pair of edges (either bad or good) leaving v. Therefore the edges are at least twice as many as the bad edges. □

Thus it is sufficient to show that in a given stage each good node is removed from the graph with constant positive probability.

Lemma 91.9 *Consider a node v in a given stage. Node v belongs to I with probability $\frac{1}{4d(v)}$.*

Proof. Let $L(v) = \{u \in N(v) \,|\, d(u) \geq d(v)\}$ be the neighbors of v of degree not smaller than $d(v)$. Then

$$Pr(v \notin I \,|\, v \in S) \leq \sum_{u \in L(v)} Pr(u \in S \,|\, v \in S) = \sum_{u \in L(v)} Pr(u \in S) \leq \sum_{u \in L(v)} \frac{1}{2d(u)} \leq \sum_{u \in L(v)} \frac{1}{2d(v)} \leq \frac{1}{2}.$$

Hence $Pr(v \in I) = Pr(v \in I \,|\, v \in S)\, Pr(v \in S) \geq \frac{1}{2}\frac{1}{2d(v)} = \frac{1}{4d(v)}$. □

Lemma 91.10 *Let v be a good node in a given stage. Node v is discarded in the stage considered with probability at least $1/36$.*

Proof. We will show that $v \in N(I) = \cup_{u \in I} N(u)$ with probability at least $1/36$. The claim follows. If v has a neighbor u of degree at most 2, by Lemma 91.9, $Pr(v \in N(I)) \geq Pr(u \in I) \geq \frac{1}{4d(u)} = \frac{1}{8}$.

Now assume all the neighbors of v have degree 3 or larger. It follows that, for every neighbor u of v, $\frac{1}{2d(u)} \leq \frac{1}{6}$. Hence by Equation 91.6 there exists a subset $M(v)$ of neighbors of v such that $\frac{1}{6} \leq \sum_{u \in M(v)} \frac{1}{2d(u)} \leq \frac{1}{3}$. Thus

$$\begin{aligned}
Pr(v \in N(I)) &\geq Pr(\exists\, u \in M(v) \cap I) \\
&\geq \sum_{u \in M(v)} Pr(u \in I) - \sum_{u,w \in M(v),\, u \neq w} Pr(u \in I \wedge w \in I) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v),\, u \neq w} Pr(u \in S \wedge w \in S) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v),\, u \neq w} Pr(u \in S) Pr(w \in S) \\
&\geq \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u \in M(v)} \sum_{w \in M(v)} \frac{1}{2d(u)} \frac{1}{2d(w)} \\
&= \left(\frac{1}{2} - \sum_{w \in M(v)} \frac{1}{2d(w)} \right) \sum_{u \in M(v)} \frac{1}{2d(u)} \geq \left(\frac{1}{2} - \frac{1}{3} \right) \frac{1}{6} = \frac{1}{36}.
\end{aligned}$$

□

91.4.2 The Distributed Maximum Matching Algorithm.

Consider an arbitrary matching M of a graph $G = (V, E)$. A node is *matched* if it is the endpoint of some edge in M, and *free* otherwise. An *augmenting path* P with respect to M is a path (of odd length) whose endpoints are free and whose edges are alternatively inside and outside M. The reason of the name is that we can obtain a matching M' of cardinality $|M| + 1$ from M, by removing from M all the edges which are also in P, and by adding to M the remaining edges of P (in other words M' is the symmetric difference $M \oplus P$ of M and P).

The algorithm by Fischer et al. is based on the following two lemmas by Hopcroft and Karp [34]. Let two paths be *independent* if they are node-disjoint. Note that a matching can be augmented along several augmenting paths simultaneously, provided that such paths are independent.

Lemma 91.11 *If a matching is augmented along a maximal set of independent shortest augmenting paths, then the shortest augmenting paths length grows.*

Lemma 91.12 *Suppose a matching M does not admit augmenting paths of length $2k - 1$ or smaller. Then the size of M is at least a fraction $\frac{k}{k+1}$ of the maximum matching size.*

Proof. Let M^* be a maximum matching. The symmetric difference $M' = M \oplus M^*$ contains $|M^*| - |M|$ independent augmenting paths with respect to M. Since each of these paths contains at least k edges of M, $|M^*| - |M| \leq |M|/k$. The claim follows. □

We are now ready to describe and analyze the approximate maximum matching algorithm by Fischer et al. The algorithm proceeds in stages. In each stage i, $i \in \{1, 2, \ldots, k\}$, the algorithm computes a maximal independent set P_i of augmenting paths of length $2i - 1$ with respect to the current matching M. Then M is augmented according to P_i. Stage i can be implemented by simulating Luby's algorithm on the auxiliary graph induced by the augmenting paths considered, where the nodes are the paths and the edges are the pairs of non-independent paths. In particular, Luby's algorithm takes $O(\log n^{2i})$ rounds in expectation in the auxiliary graph, where each such round can be simulated within $O(i)$ rounds in the original graph. Note that, by Lemma 91.11, at the end of stage i there are no augmenting paths of length $2i - 1$ or smaller. It follows from Lemma 91.12 that at the end of the k-th stage the matching computed is $\frac{k}{k+1}$-approximate. The total expected number of rounds is trivially $O(k^3 \log n)$. The following theorem summarizes the discussion above.

Theorem 91.1 *For every integer $k > 0$, there is a distributed algorithm which computes a matching of cardinality at least $\frac{k}{k+1}$ times the maximum matching cardinality within $O(k^3 \log n)$ communication rounds in expectation.*

Wattenhofer and Wattenhofer [35] gave a $O(\log^2 n)$ randomized algorithm to compute a constant approximation in the weighted case. In the deterministic case weaker results are available. This is mainly due to the fact that we are not able to compute maximal independent sets deterministically. Hańćkowiak, Karoński, and Panconesi [36, 37] described an efficient distributed deterministic algorithm to compute a maximal matching. Recall that any maximal matching is a 2-approximation for the maximum matching problem. Recently a 1.5 deterministic distributed approximation algorithm was described in [38].

91.5 LP-based Distributed Algorithms

It might come as a surprise that LP-based methods find their application in a distributed setting. In this section we describe some primal-dual algorithms for vertex cover problems that give "state-of-the-art" approximations. In general it seems that the primal-dual method, one of the most successful techniques in approximation algorithms, when applied to graph algorithms exhibits "local" properties that makes it amenable to a distributed implementation. The best way to explain what we mean is to work out an example.

We will illustrate the method by considering the *vertex cover* problem: given an undirected graph $G = (V, E)$, with positive weights $\{c(v)\}_{v \in V}$, compute a minimum cost subset V' of nodes such that each edge is incident to at least one node in V'. This NP-hard problem is approximable within 2 [39], and not approximable within 1.1666 unless P=NP [40]. In the centralized case there is a primal-dual 2-approximation algorithm. The distributed implementation we give yields a $2 + \epsilon$ approximation, where ϵ can be any fixed constant greater than zero. The number of communication rounds of the algorithm is $O(\log n \ \log \frac{1}{\epsilon})$.

The sequential primal-dual algorithm works as follows. We formulate the problem as an integer program (IP):

$$\min \sum_{v\in V} c(v)\cdot x_v \tag{IP}$$

$$\text{s.t} \quad x_v + x_u \geq 1 \qquad \forall e=(u,v)\in E \tag{91.7}$$

$$x_v \in \{0,1\} \qquad \forall v \in V \tag{91.8}$$

The binary indicator variable x_v, for each $v \in V$, takes value one if $v \in V'$, and zero otherwise.

We now let (LP) be the standard LP relaxation obtained from (IP) by replacing the constraints (91.8) by $x_v \geq 0$ for all $v \in V$. In the linear-programming dual of (LP) we associate a variable α_e with constraint (91.7) for every $e \in E$. The linear programming dual (D) of (LP) is then

$$\max \sum_{e\in E} \alpha_e \tag{D}$$

$$\text{s.t} \quad \sum_{e=(u,v)\in E} \alpha_e \leq c(v) \qquad \forall v \in V \tag{91.9}$$

$$\alpha_e \geq 0 \qquad \forall e \in E \tag{91.10}$$

The starting primal and dual solutions are obtained by setting to zero all the variables x_v and α_e. Observe that the dual solution is feasible while the primal one is not. We describe the algorithm as a continuous process. We let all the variables α_e grow at uniform speed. As soon as one constraint of type (91.9) is satisfied with equality (it becomes *tight*), we set the corresponding variable x_v to one, and we freeze the values α_e of the edges incident to v. The α-values of frozen edges do not grow more, so that the constraint considered remains tight. The process continues until all edges are frozen. When this happens the primal solution becomes feasible. To see why, suppose not. But then there is an edge $e = uv$ which is not covered, i.e. $x_u = x_v = 0$. This means that the constraints corresponding to u and v are not tight and α_e can continue to grow, a contradiction.

Thus the set $V' := \{u : x_u = 1\}$ is a cover. Its cost is upper-bounded by twice the cost of the dual solution:

$$\sum_{v\in V} c(v)\, x_v = \sum_{v\in V'} c(v) \leq \sum_{v\in V'} \sum_{e=(u,v)\in E} \alpha_e \leq 2\sum_{e\in E} \alpha_e.$$

Thus the solution computed is 2-approximate by weak duality.

The continuous process above can be easily turned into a discrete one. Let $c'(v)$ be the difference between the right-hand side and the left-hand side of constraints (91.9) in a given instant of time (*residual weight*):

$$c'(v) = c(v) - \sum_{e=(u,v)\in E} \alpha_e.$$

Let moreover $d'(v)$ be the current number of non-frozen (*active*) edges incident to v. The idea is to raise in each step the dual value α_e of all the active edges by the minimum over all nodes v such that $x_v = 0$ of the quantity $c'(v)/d'(v)$. This way, in each step at least one extra node enters the vertex cover.

There is a simple-minded way to turn the algorithm above into a distributed algorithm: each node v maintains the quantities $c'(v)$ and $d'(v)$. A node is *active* if $c'(v) > 0$ and $d'(v) > 0$, that is if v and at least one of its neighbors are not part of the vertex cover. In each round each active node v sends a *proposal* $c'(v)/d'(v)$ to all its active neighbors. Then it decreases $c'(v)$ by the minimum of all the proposals sent and received. If $c'(v)$ becomes zero, v enters the vertex cover. Otherwise, if $d'(v)$ becomes zero, v halts since all its neighbors already belong to the vertex cover.

The main drawback of this approach is that it is very slow. In fact, it may happen that in each step a unique node enters the vertex cover, thus leading to a linear number of rounds. Khuller, Vishkin, and Young [41] showed how to circumvent this problem by loosing something in the approximation. Here we will present a simplified version of their algorithm and analysis (which was originally thought for weighted set cover in a parallel setting). The idea is to slightly relax the condition for a node v to enter the vertex cover: it is sufficient that the residual weight $c'(v)$ falls below $\epsilon\, c(v)$, for a given (small) constant $\epsilon > 0$.

Theorem 91.2 *The algorithm above computes a $\frac{2}{1-\epsilon}$-approximate vertex cover within $O(\log n \, \log \frac{1}{\epsilon})$ rounds.*

Proof. The bound on the approximation easily follows by adapting the analysis of the primal-dual centralized approximation algorithm:

$$(1-\epsilon)\, apx = \sum_{v\in V'} (1-\epsilon)\, c(v) \le \sum_{v\in V'} \sum_{e=(u,v)\in E} \alpha_e \le 2 \sum_{e\in E} \alpha_e \le 2\, opt.$$

In order to bound the number of rounds we use a variant of the notion of good nodes introduced in Section 91.4.1. Consider the graph induced by the active nodes in a given round, and call the corresponding edges *active*. Let us direct all the active edges toward the endpoint which makes the smallest proposal. A node is *good* if its in-degree is at least one third of its (total) degree. By basically the same argument as in Section 91.4.1, at least one half of the edges are incident to good nodes. Moreover, the residual weight of a node which is good in a given round decreases by at least one third in the round considered. As a consequence, a node can be good in at most $\log_{3/2}\frac{1}{\epsilon}$ rounds (after those many rounds it must enter the vertex cover).

We will show next that the total number of active edges halves every $O(\log\frac{1}{\epsilon})$ rounds by means of a potential function argument. It follows that the total number of rounds is $O(\log m\,\log\frac{1}{\epsilon}) = O(\log n\,\log\frac{1}{\epsilon})$. Let us associate $2\log_{3/2}\frac{1}{\epsilon}$ credits to each edge, and thus $2m\log_{3/2}\frac{1}{\epsilon}$ credits to the whole graph. When a node v is good in a given step, we remove one credit from each edge incident to it. Observe that an active edge e in a given round must have at least 2 credits left. This is because otherwise one of the endpoints of e would already belong to the vertex cover, and thus e could not be active. By m_j we denote the number of active edges in round j. Recall that in each round at least one half of the edges are incident to a good node, and such edges loose at least one credit each in the round considered. Thus the total number of credits in round j decreases by a quantity g_j which satisfies $g_j \geq m_j/2$. Consider an arbitrary round i, and let k be the smallest integer such that $m_{i+k} < m_i/2$ (or $i+k$ is the last round). Is is sufficient to show that $k = O(\log\frac{1}{\epsilon})$. In each round j, $j \in \{i, i+1, \ldots, i+k-1\}$, the number of edges satisfies $m_j \geq m_i/2$. The total number of credits at the beginning of round i is at most $2m_i\log_{3/2}\frac{1}{\epsilon}$, and the algorithm halts when no credit is left. Therefore

$$2m_i\log_{3/2}\frac{1}{\epsilon} \geq \sum_{j=i}^{i+k-1} g_j \geq \sum_{j=i}^{i+k-1}\frac{m_j}{2} \geq \sum_{j=i}^{i+k-1}\frac{m_i}{4} = k\frac{m_i}{4} \quad\Rightarrow\quad k \leq 8\log_{3/2}\frac{1}{\epsilon} = O(\log\frac{1}{\epsilon}).$$

□

By choosing $\epsilon = 1/(nC+1)$, where C is the maximum weight, the algorithm by Khuller et al. computes a 2-approximate vertex cover within $O(\log n\,\log(nC))$ rounds. Recently Grandoni, Könemann, and Panconesi [42] showed how to achieve the same task in $O(\log(nC))$ rounds by means of randomization. They reduce the problem to the computation of a maximal matching in an auxiliary graph of nC nodes (to have an idea of the reduction, see Figure 91.2). Such matching can be computed in $O(\log(nC))$ rounds via the randomized, distributed maximal matching algorithm by Israeli and Itai [43]. The authors also show how to keep small the message size and the local computation time by computing the matching *implicitly*.

The *capacitated* vertex cover problem is the generalization of the vertex cover problem where each node v can cover only a limited number $b(v) \leq d(v)$ of edges incident to it. Grandoni, Könemann, Panconesi, and Sozio [44] showed how to compute within $O(\frac{\log nC}{\epsilon})$ rounds an $(2+\epsilon)$-approximate solution, if any, which violates the capacity constraints by a factor at most $(4+\epsilon)$. They also proved that any distributed constant approximation algorithm must violate the capacity constraints by a factor at least 2. This, together with the known lower bounds on the approximation of (classical) vertex cover, shows that their algorithm is the best possible modulo constants. The algorithm by Grandoni et al. builds up on a primal-dual centralized algorithm developed for the purpose, which computes a 2 approximation with a factor 2 violation of the capacity constraints. Turning such primal-dual algorithm into a distributed protocol is far more involved than in the case of classical vertex cover.

91.6 What Can and Cannot be Computed Locally?

This fundamental question in distributed computing was posed by Moni Naor and Larry Stockmeyer [45]. Here, "locally" means that the nodes of the network use information available locally from a neighborhood that can be reached in time much smaller than the size of the network. For many natural distributed network problems such as leader election and consensus the parameter determining the time complexity is not the number of vertices, but the network *diameter* D which is the maximum distance (number of hops) between any two nodes [46]. A natural question is whether other fundamental primitives can be computed in $O(D)$ time in a distributed setting. If the model allows messages of unbounded size, then there is a trivial affirmative answer to this question: collect all the information at one vertex, solve the problem locally and then transmit the result to all vertices. The problem is therefore only interesting in the more realistic model where we assume that each link can transmit only B bits in any time step (B is usually taken to be a constant or $O(\log n)$).

A landmark negative result in this direction was that of Nati Linial [47] which investigated the time complexity of various global functions of a graph computed in a distributed setting. Suppose that n processors are arranged in a ring and can communicate only with their immediate neighbors. Linial showed that a 3-coloring of the n-cycle requires time $\Omega(\log^* n)$. This result was extended to randomized algorithms by Moni Naor [48]: any probabilistic algorithm for 3-coloring the ring must take a least $\frac{1}{2}\log^* n - 2$ rounds, otherwise the probability that all processors are colored legally is less than $\frac{1}{2}$. The bound is tight (up to a constant factor) in light of the deterministic algorithms of R. J. Cole and U. Vishkin [49].

There has been surprisingly little continuation of work in this direction until fairly recently. Garay, Kutten and Peleg [50] gave an algorithm of complexity $O(D + \sqrt{n}\log n)$ to compute a minimum spanning tree (MST) of a graph on n vertices with diameter D. Similar bounds were attained by other methods, but none managed to break the $\sqrt{n}$ barrier, leading to the suspicion that it might be impossible to compute the MST in time $o(\sqrt{n})$ and so this problem is fundamentally harder than the other paradigm problems. The issue was finally settled in [51] who showed a $\Omega(\sqrt{n})$ lower bound on the problem (up-to log factors). Subsequently, Elkin [52] improved the lower bound and also extended it to distributed approximation algorithms. Kuhn, Moscibroda and Wattenhofer [53] gave lower bounds on the complexity of computing the minimum vertex cover (MVC) and the minimum dominating set (MDS) of a graph: in k communication rounds, the MVC and MDS can only be approximated to factors of $\Omega\left(n^{ck^2}/k\right)$ and $\Omega\left(\Delta^{1/k}/k\right)$ (where Δ is the maximum degree of the graph). Thus, the number of rounds required to reach a constant or even a poly-log approximation is at least $\Omega\left(\sqrt{\log n/\log\log n}\right)$ and $\Omega\left(\log\Delta/\log\log\Delta\right)$. The same lower bounds also apply to the construction of maximal matchings and maximal independent sets via a simple reduction.

91.6.1 A Case Study: Minimum Spanning Tree

Here,we give a self-contained exposition of the lower bound for the MST problem due to [51, 52]. We will give the full proof of a bound somewhat weaker than the optimal result of Elkin to convey the underlying ideas more clearly. The basic idea is easy to explain using the example of Peleg and Rubinovich [51], see Fig. 91.3. The network consists of m^2 country road and one highway. Each country road has m toll stations and between every two successive toll station are m towns. The highway has m toll stations with no towns in between. Each toll station number i on each country road is connected to the corresponding highway toll station. The left end of country road i is labelled s_i and its right end r_i. The left end of the highway is labelled s and the right end r. This is the basic underlying graph. Note that there are $\Theta(m^4)$ vertices and the diameter is $\Theta(m)$.

As for the weights, every edge along the highway or on the country roads has weight 0. The roads connecting the toll stations on the country roads to the corresponding toll stations on the highway have weight ∞ *except for the first and last toll stations*. The toll station connections on the right end between each r_i and r are all 1. At the left end, between each s_i and s, they take either the value 0 or ∞.

What does the MST of this network look like? First, we may as well include the edges along the highway and each path since these have zero cost. Also the intermediate connecting edges have weight ∞ and so are excluded. That leaves us with the connecting edges on the left and on the right. The choice here depends on the weights on the left connecting edges. There are m connecting edges from the left vertex s. If the edge (s, s_i) has weight ∞, then we must exclude this and include the matching connection (r_i, r) at the right end. On the other hand, if edge (s, s_i) has weight 0, then we must include this and exclude the corresponding edge (r_i, r) at the right to r. Thus there are m^2 decisions made at s depending on the weights of the corresponding edges, and these decision must be conveyed to r to pick the corresponding complementary edges. How quickly can these m^2 bits be conveyed from s to r? Clearly it would take very long to route along the country roads, and so one must use the highway edges instead. Each highway edge can forward only B bits at any time step. So, heuristically, transporting the m^2 bits takes $\Omega(m^3/B)$ steps.

To make this heuristic argument formal, Peleg and Rubinovich introduced a *mailing problem* to be solved on a given network . In the example above, the *sender* s has m^2 bits that need to be transported to the *receiver* r. At each step one can forward B bits along any edge. How many steps do we need to correctly route the m^2 bits from the sender to the receiver? It is easy to see that there is a reduction from the mailing problem to that of computing the MST: for each of the input bits at s, set the weights on the connecting edges accordingly: the weight (s, s_i) is ∞ if the input bit i is 1 and 0 otherwise. Now compute the MST. Then, if vertex r notices that the edge (r_i, r) is picked in the MST, it decodes i as 1 and as 0 otherwise. This will correctly solve the mailing problem, due to the structure of the MST discussed above. Thus, a lower bound on the mailing problem implies the same lower bound on the MST problem.

In fact by a slight change, the correspondence can be extended from exact to approximation algorithms. Elkin [52] introduced the *corrupted mail* problem. Here there are Γ bits at the sender exactly $\alpha\Gamma$ of which are 1's, where α and Γ are parameters. In the example above, $\Gamma = m^2$. The receiver should get Γ bits delivered to it, but these are allowed to be somewhat corrupted. The restrictions are (a) any input bit that was 1 must be transmitted correctly without corruption and (b) the total number of 1's delivered can be at most $\beta\Gamma$ where $\beta \geq \alpha$ is another parameter. Consider solving the (α, β) corrupted mail problem on the Peleg-Rubinovich example. As in the reduction before, the vertex s sets the weights on the left connections according to its input and so exactly $\alpha\Gamma$ connections have weight ∞, and the rest 0. The optimal MST has weight exactly $\alpha\Gamma$ obtained by picking the corresponding right connections. Now, instead of the optimal MST, suppose we apply a protocol to compute a β/α approximation. This approximate MST can have weight at most $\beta\Gamma$, and it must include the connection edges at r paired with the infinite weight edges at s. Thus, if r sets its bits as before corresponding to which of its connections are in the approximate MST, we get a correct protocol for the (α, β) corrupted mail problem. Thus a lower bound for the (α, β) corrupted mail problem implies the same lower bound for a $\frac{\beta}{\alpha}$ approximate MST.

We are thus left with the task of proving a lower bound for the corrupted mail problem. Let the *state* $\psi(v,t)$ of a vertex v at some time t denote the sequence of messages it has received up to this time. Consider the start vertex s at time 0: this can be in any of $\binom{\Gamma}{\alpha\Gamma}$ states corresponding to the input it receives. At this time, on the other hand, the vertex r (and indeed, any other vertex) is in a fixed state (having received no messages at all). As time progresses and messages are passed, the set of possible states that other vertices are in expands. Eventually, the set of possible states that vertex r is in must be large enough to accommodate the output corresponding to all the possible inputs at s. Each possible state of r with at most $\beta\Gamma$ 1s can be the correct answer to at most $\binom{\beta\Gamma}{\alpha\Gamma}$ input configurations at s. Hence, the set of output states at r must be at least $\binom{\Gamma}{\alpha\Gamma}/\binom{\beta\Gamma}{\alpha\Gamma} \geq (1/e\beta)^{\alpha\Gamma}$.

Now, we will argue that it must take a long time for any protocol, before enough messages arrive at r for the set of its possible states to have this size. Consider the *tail sets* $T_i, i \geq 1$ which consist of the tail of each country road from vertex i until the end, and the corresponding fragment of the highway consisting of the vertices $h_{\lceil i/m\rceil m}$ until h_{m^2}. Also, set $T_0 := V \setminus \{h_0\}$. For a subset of vertices U, let $\mathcal{C}(U,t)$ denote set of all possible vectors of states of the vertices in U at time t, and let $\rho(U,t) := |\mathcal{C}(U,t)|$. Note that $\rho(T_0,0) = 1$ although $\rho(\{s\},0) = \binom{\Gamma}{\alpha\Gamma}$.

We now focus on how set of configurations of the tail sets T_i grow in time. Fix a configuration $C \in \mathcal{C}(T_t,t)$. How many configurations in $\mathcal{C}(T_{t+1},t+1)$ can this branch into? The tail set T_{t+1} is connected to the rest of the graph by one highway edge f and by m^2 path edges. Each of the path edges carries a unique message determined by the state of the left end point in configuration C. The state of the left end point of the highway edge f is not determined by C and hence there could be a number of possible messages that could be relayed along it. However, because of the restriction that at most B bits can be transmitted along an edge at any time step, the total number of possible behaviors observable on edge f at this time step is at most 2^B+1. Thus the configuration C can branch off into at most 2^B+1 possible configurations $C' \in \mathcal{C}(T_{t+1},t+1)$. Thus we have argued that for $0 \leq t < m^2$, $\rho(T_{t+1},t+1) \leq (2^B+1)\rho(T_t,t)$. By induction, this implies that for $0 \leq t < m^2$, $\rho(T_t,t) \leq (2^B+1)^t$. Thus finally, we have, that if t^* is the time at which the protocol ends, then either $t^* \geq m^2$, or $(1/e\beta)^{\alpha\Gamma} \leq \rho(\{r\},t^*) \leq \rho(T_{t^*},t^*) \leq (2^B+1)^{t^*}$. Hence, $t^* \geq \min\left(m^2, \alpha\Gamma\log(\frac{1}{e\beta})/(B+1)\right)$.

Recalling that $\Gamma = m^2$ in our specific graph, and taking β to be a constant such that $\beta e < 1$, $t^* = \Omega(\alpha m^2/B)$, or, in terms of the number of vertices $n = \Theta(m^4)$ of the graph, $t^* = \Omega(\alpha\sqrt{n}/B)$. If we have a $H := \beta/\alpha$ approximation algorithm for the MST, this implies that $t^* = \Omega(\sqrt{n}/HB)$, implying the trade-off $t^*H = \Omega(\sqrt{n}/B)$ between time and approximation. Elkin [52] improves the lower bound for t^* to $t^* = \Omega\left(\sqrt{n/B}/H\right)$, implying the time-approximation tradeoff $t^{*2}H = \Omega\left(\sqrt{n/B}\right)$, and gives a protocol achieving this tradeoff.

91.6.2 The Role of Randomization in Distributed Computing

Does randomization help in a distributed setting? This is a fundamental open question in distributed computing. For some of the problems discussed, such as 3-coloring on a ring, we have noted that matching lower bounds hold for randomized algorithms. By the usual application of Yao's Minimax Theorem, Elkin's lower bound also applies to randomized algorithms. For the problem of computing maximal matchings and maximal independent sets, there are simple randomized algorithms, whereas the result of Kuhn et al [53] shows a super-poly-log lower bound for deterministic algorithms. A classification of problems by the degree to which randomization helps is an interesting open problem.

References

[1] Basagni, S., Mastrogiovanni, M., Panconesi, A., and Petrioli, C., Localized protocols for ad hoc clustering and backbone formation: A performance comparison, *IEEE Trans. on Parallel and Dist. Comput.*, (to appear).

[2] Rajagopalan, S. and Vazirani, V.V., Primal-dual RNC approximation algorithms for set cover and covering integer programs, *SIAM J. on Comput.*, 28(2), 525(electronic), 1999.

[3] Vazirani, V.V., *Approximation algorithms*, Springer-Verlag, Berlin, 2001.

[4] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation*, Springer-Verlag, Berlin, 1999.

[5] Raz, R. and Safra, S., A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, in *Proc. of STOC*, 1997, 475.

[6] Arora, S. and Sudan, M., Improved low-degree testing and its applications, *Combinatorica*, 23(3), 365, 2003.

[7] Feige, U., A threshold of $\ln n$ for approximating set cover, *JACM*, 45(4), 634, 1998.

[8] Dubhashi, D., Mei, A., Panconesi, A., Radhakrishnan, J., and Srinivasan, A., Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons, in *Proc. of SODA*, 2003, 717.

[9] Jia, L., Rajaraman, R., and Suel, T., An efficient distributed algorithm for constructing small dominating sets, *Dist. Comput.*, 15, 193, 2002.

[10] Kuhn, F. and Wattenhofer, R., Constant-time distributed dominating set approximation, *Dist. Comput.*, 17(4), 2005.

[11] Karp, R.M., Probabilistic recurrence relations, *JACM*, 41(6), 1136, 1994.

[12] Dubhashi D. and Panconesi, A., Concentration of measure for the analysis of randomised algorithms, manuscript, 2005.

[13] Matoušek. J., *Lectures on Discrete Geometry*, Graduate Texts in Math., 212, Springer, 2002.

[14] Bellare, M., Goldreich, O., and Sudan, M., Free bits, pcps and non-approximability - towards tight results, *SIAM J. on Comput.*, 27, 804, 1998.

[15] Feige, U. and Kilian, J., Zero knowledge and the chromatic number, *JCSS*, 57, 187, 1998.

[16] Halldòrsson, M.M., A still better performance guarantee for approximate graph coloring, *Inf. Proc. Lett.*, 45, 19, 1993.

[17] Johansson, O., Simple distributed $\delta + 1$-coloring of graphs, *Inf. Proc. Lett.*, 70(5), 229, 1999.

[18] Luby, M., Removing randomness in parallel computation without a processor penalty, *JCSS*, 47(2), 250, 1993.

[19] Finocchi, I., Panconesi, A., and Silvestri, R., An experimental study of simple, distributed vertex colouring algorithms, in *Proc. of SODA*, 2002.

[20] Czygrinow, A., Hańćkowiak, M., and Karoński, M., Distributed O(Delta log(n))-edge-coloring algorithm, in *Proc. Eur. Symp. on Algorithms*, 2001, 345.

[21] Bollobas, B., *Graph Theory: An Introductory Course*, Springer-Verlag, 1979.

[22] Dubhashi, D., Grable, D., and Panconesi, A., Nearly-optimal, distributed edge-colouring via the nibble method, *Theor. Comp. Sci.*, 203, 225, 1998.

[23] Grable, D. and Panconesi, A., Nearly optimal distributed edge colouring in O$(\log\log n)$ rounds, *Random Struct. and Algorithms*, 10(3), 385, 1997.

[24] Bollobas, B., Chromatic number, girth and maximal degree, *SIAM J. on Disc. Math.*, 24, 311, 1978.

[25] Grable, D. and Panconesi, A., Fast distributed algorithms for brooks-vizing colourings, in *Proc. of SODA*, 1998, 473.

[26] McDiarmid, C., Concentration, in *Probabilistic methods for algorithmic discrete mathematics*, Algorithms Combin., 16, 1998, 195.

[27] Molloy, M. and Reed, B., A bound on the strong chromatic index of a graph, *J. of Comb. Theory*.

[28] Mitzenmacher, M. and Upfal, E., *Probability and Computing*, Cambridge University Press, 2005.

[29] Dubhashi, D.P., Martingales and locality in distributed computing, in *Foundations of Software Tech. and Theor. Comp. Sci.*, 174. 1998.

[30] Fischer, T., Goldberg, A.V., Haglin, D.J., and Plotkin, S., Approximating matchings in parallel, *Inf. Proc. Lett.*, 46, 115, 1993.

[31] Luby, M., A simple parallel algorithm for the maximal independent set problem, in *Proc. of STOC*, 1985, 1.

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

[33] Kozen, D., *The Design and Analysis of Algorithms*, Springer-Verlag, 1992.

[34] Hopcroft, J.E. and Karp, R.M., An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, *SIAM J. on Comput.*, 2, 225, 1973.

[35] Wattenhofer, M. and Wattenhofer, R., Distributed weighted matching, in *Proc. Intl. Symp. on Dist. Comput.*, 2004, 335.

[36] Hańćkowiak, M., Karoński, M., and Panconesi, A., On the distributed complexity of computing maximal matchings, in *Proc. of SODA*, 1998, 219.

[37] Hańćkowiak, M., Karoński, M., and Panconesi, A., On the distributed complexity of computing maximal matchings, *SIAM J. on Disc. Math.*, 15(1), 41, 2001.

[38] Czygrinow, A., Hańćkowiak, M., and Szymanska, E., A fast distributed algorithm for approximating the maximum matching, in *Proc. Eur. Symp. on Algorithms*, 2004, 252.

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

[40] Håstad, J., Some optimal inapproximability results, in *Proc. of STOC*, 1997, 1.

[41] Khuller, S., Vishkin, U., and Young, N., A primal-dual parallel approximation technique applied to weighted set and vertex cover, *J. of Algorithms*, 17(2), 280, 1994.

[42] Grandoni, F., Könemann, J., and Panconesi, A., Distributed weighted vertex cover via maximal matchings, in *Intl. Comput. and Comb. Conf.* 2005.

[43] Israeli, A. and Itai, A., A fast and simple randomized parallel algorithm for maximal matching, *Inf. Proc. Lett.*, 22, 77, 1986.

[44] Grandoni, F., Könemann, J., Panconesi, A., and Sozio, M., Primal-dual based distributed algorithms for vertex cover with semi-hard capacities, in *Symp. on Principles of Dist. Comput.*, 2005, 118.

[45] Nar, M. and Stockmeyer, L., What can be computed locally? *SIAM J. on Comput.*, 24(6), 1259, 1995.

[46] Peleg, D., *Distributed computing*, SIAM Monographs on Disc. Math. and Appl., 5, 2000.

[47] Linial, N., Locality in distributed graph algorithms, *SIAM J. on Comput.*, 21(1), 193, 1992.

[48] Naor, M., A lower bound on probabilistic algorithms for distributive ring coloring, *SIAM J. on Disc. Math.*, 4(3), 409, 1991.

[49] Cole, R. and Vishkin, U., Deterministic coin tossing with applications to optimal parallel list ranking, *Inf. and Control*, 70(1), 32, 1986.

[50] Garay, J.A., Kutten, S., and Peleg, D., A sublinear time distributed algorithm for minimum-weight spanning trees, *SIAM J. on Comput.*, 27(1), 302, 1998.

[51] Peleg, D. and Rubinovich, V., A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction, *SIAM J. on Comput.*, 30(5), 1427, 2000.

[52] Elkin, M., Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem, in *Proc. STOC*, 2004, 331.

[53] Kuhn, F., Moscibroda, T., and Wattenhofer, R., What cannot be computed locally! in *Proc. Symp. on Principles of Dist. Comput.*, 2004.

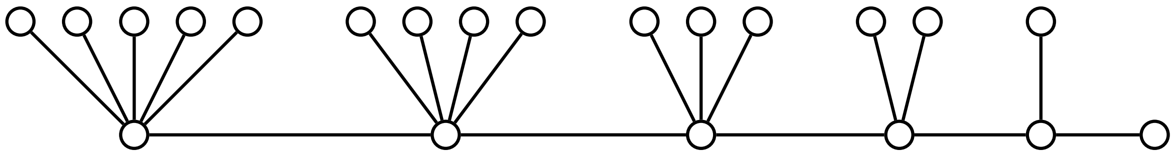

Figure 91.1: Example of lower bound graph for $k = 6$. The number of nodes is $n = k(k+1)/2 = \Theta(k^2)$. The bottom nodes are selected by greedy, one by one from left to right. The number of rounds is $k - 1$.

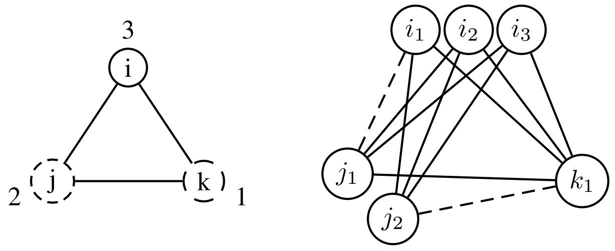

Figure 91.2: A weighted graph G (on the left) with the corresponding auxiliary graph $\widetilde{G}$. A maximal matching M of $\widetilde{G}$ is indicated via dashed lines. The nodes of G such that all the corresponding nodes in $\widetilde{G}$ are matched form a 2-approximate vertex cover.

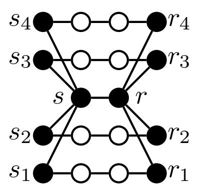

Figure 91.3: MST lower bound graph for $m = 2$. The black nodes are the toll stations and the white nodes are the towns.

Index