Local Labeling and Resource Allocation Using Preprocessing*

Hagit Attiya† Hadas Shachnai‡ Tami Tamir§

Department of Computer Science
The Technion, Haifa 32000, Israel

June 24, 1996

Abstract

This paper studies the power of non-restricted preprocessing on a communication graph G, in a synchronous, reliable system. In our scenario, arbitrary preprocessing can be performed on G, after which a sequence of labeling problems have to be solved on different subgraphs of G. We suggest a preprocessing that produces an orientation of G. The goal is to exploit this preprocessing for minimizing the radius of the neighborhood around each vertex from which data has to be collected in order to determine a label. We define a set of labeling problems for which this can be done. The time complexity of labeling a subgraph depends on the topology of the graph G and is always less than $\min\{\chi(G), O((\log n)^2)\}$. On the other hand, we show the existence of a graph for which even unbounded preprocessing does not allow fast solution of a simple labeling problem. Specifically, it is shown that a processor needs to know its $\Omega(\log n/\log\log n)$-neighborhood in order to pick a label.

Finally, we derive some results for the resource allocation problem. In particular, we show that $\Omega(\log n/\log\log n)$ communication rounds are needed if resources are to be fully utilized. In this context, we define the *compact coloring* problem, for which the orientation preprocessing provides fast distributed labeling algorithm. This algorithm suggests efficient solution for the resource allocation problem.

Key words: locality, preprocessing, orientation, labeling, resource allocation, response time.

---

*A preliminary version of this paper appeared in proceedings of *the 8th International Workshop on Distributed Algorithms,* Terschelling, The Netherlands, September/October 1994, (G. Tel and P. Vitanyi, Eds.), pp. 194–208, Lecture Notes in Computer Science #857, Springer-Verlag. This work was supported by grant No. 92-0233 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, by the fund for the promotion of research in the Technion, and by Technion VPR funds.

†hagit@cs.technion.ac.il

‡hadas@cs.technion.ac.il. Part of this work was done while the author was at IBM T.J. Watson Research Center, Yorktown Heights, NY.

§ttamir@iil.intel.com

1 Introduction

The time required to perform certain computations in message-passing systems depends, in many cases, on the *locality* of information, i.e., the distance to which information should be forwarded. Clearly, within t communication rounds, a processor can get information only from processors located within distance t. The study of problems that are local, i.e., in which the value of a processor depends only on its close-by neighborhood, has attracted much attention, e.g., [13, 16, 12, 18, 11]. This study assumed that processors have no knowledge about the network topology. In many common scenarios, this is not the situation: If the same problem has to be solved many times on different sub-networks of a fixed network G, then it might be worthwhile to conduct some preliminary preprocessing on G.

We study *labeling problems*, in which each processor has to pick a label, subject to some restrictions on the labeling of the whole network. We allow arbitrary preprocessing on G. Afterwards, several instances of the same labeling problem need to be solved on different sub-networks G' of G. All processors of G can participate in the algorithm when a particular sub-network G' is labeling itself, but only the processors of G' have to pick labels. It is assumed that the system is synchronous and operates in rounds; there is no bound on message length, and local computation is unlimited. Furthermore, we assume the system is completely reliable. The preprocessing attempts to increase the locality of the problem, i.e., decrease the radius of the neighborhood a processor v needs to know in order to pick a label.

The preprocessing we present produces an orientation that assigns priorities to the processors. Later, when a processor has to compute its label in some subgraph G', it only considers processors with higher priorities. We define a parameter that quantifies the quality of these orientations, denoted by $t(G)$. $t(G)$ depends on the topology of G, and it is always less than $\min\{\chi(G), O((\log n)^2)\}$.

We define *extendible labeling problems*, in which a labeled graph can be extended by an independent set of vertices to a larger labeled graph without invalidating the original labels. The maximal independent set problem and the $(\Delta+1)$-coloring problem are extendible. We suggest an efficient preprocessing on G which allows to solve these problems within $t(G)$ rounds on any subgraph of G. We also discuss a distributed randomized preprocessing on G that takes $O((\log n)^2)$ rounds and enables to solve these problems on any subgraph of G within $O((\log n)^2)$ rounds. This gives a distributed randomized algorithm for *compact coloring*. Bar-Noy et al. ([6]) have shown that this algorithm provides efficient solutions to the resource allocation problem, for a large class of graphs.

We introduce a problem in which processors have to communicate with processors at a non-constant distance, even after unbounded preprocessing. The problem is *k-dense coloring*, which is a restricted coloring problem. A coloring is k-dense if every vertex with color $c > k$ should have a neighbor with color c', $c-k \leq c' \leq c-1$. Note that validating that a coloring is k-dense requires only checking with the neighbors (i.e., processors that are at distance 1). We prove that there exists a network on which processors must know their $\Omega(\log\, n/\log\log\, n)$-neighborhood

in order to pick a color. That is, for some networks, even unbounded preprocessing does not allow to solve the problem locally.

The locality of distributed computations was first studied by Cole and Vishkin, who showed in [9] that a 3-coloring of a ring requires only the knowledge of a $O(\log^* n)$-neighborhood; this bound was shown to be tight by Linial [12]. The more general problem of computing labels locally was studied by Naor and Stockmeyer [16] in the case where no preprocessing is allowed. They present local algorithms for some labeling problems whose validity can be checked locally, and also show that randomization does not help in making a labeling problem local. In follow-up work, Mayer, Naor and Stockmeyer [15] consider the amount of initial symmetry-breaking needed in order to solve certain labeling problems.

Other, less related, works studied coloring and the maximal independent set problem in graphs (e.g., Goldberg, Plotkin and Shannon [11], Szegendy and Vishwanathan [18], and Panconesi and Srinivasan [17]). Another use of graph-theoretic techniques for local algorithms appears in works on sparse partition [2, 14]. In these works, preprocessing is applied in order to partition a graph into graphs with small diameters. Given such a partition, it is possible to solve the problem locally for each sub-graph and then compose the resulting labels. See also the survey by Linial [13], which describes other works on locality in distributed computation.

Preprocessing is very helpful in the context of on-going problems, such as *resource allocation* [7], where jobs with conflicting resource requirements have to be scheduled efficiently. An instance of the problem is a *communication graph* G. The vertices represent processors, and there is an edge between a pair of processors if they may compete on a resource. The resource requirements of a processor may vary, and current requirements are represented by a dynamic *conflict graph* C, where the vertices are processors waiting to execute their jobs, and there is an edge between two processors that currently compete on some resource. (Note that $C \subseteq G$.)

We consider a restricted version of the resource allocation problem: A schedule is *k-compact*, if for every waiting processor p_i, in every k rounds, either p_i runs, or there exists some conflicting neighbor of p_i which runs. This guarantees that p_i is delayed only because one of its conflicting neighbors is running.

The lower bound for the k-dense coloring problem implies that there is no preprocessing which enables a distributed k-compact schedule within less than $\Omega(\log n / \log\log n)$ rounds. We present a distributed algorithm which is μ-compact, where μ is a known upper bound on the execution time of a job; the algorithm uses preprocessing that produces a t-orientation. The response time of our algorithm is $\delta_i\mu + t(G)$, with δ_i the degree of p_i in C.

The resource allocation problem was introduced by Chandy and Misra [7]. In their definition, known as the *dining philosophers* problem, the resource requirements of the processors are static. We consider the dynamic version of the problem, known as the *drinking philosophers* problem. Several algorithms for the drinking philosophers problem are known. Without preprocessing, the best algorithm to date [5] achieves $O(\delta_i\mu + \delta \log n)$ response time, where δ_i is the degree of p_i in C and δ is the maximal degree in C. In contrast, by using preprocessing, our algorithm achieves a response time of $\delta_i\mu + t(G)$. An algorithm that relies on preprocessing

(which colors the communication graph to induce priorities between processors) and achieves a response time of $O(\delta^2\mu)$ was presented in [8].

The usage of a preprocessing that induces an orientation of the conflict graph was first considered in [7]; Barbosa and Gafni [4] present theoretical results concerning the maximal concurrency which may be achieved using orientation. Like our algorithms, in these papers, the orientation is used to induce priorities between processors, so as to decrease the waiting time of processors. However, in this work the quality of a graph orientation is measured as the maximal *directed* length in the graph, which corresponds to the maximal *waiting chain* for a particular processor. In contrast, our measure for the quality of an orientation is the maximal *undirected* distance between two processors that are connected by a directed path. This allows us to combine the orientation preprocessing with a local distributed labeling algorithm, such that the resulting waiting time for each processor is bounded by a small constant, although the length of the maximal directed path may equal to the size of the graph.

The rest of this paper is organized as follows. In Section 2 we give some basic definitions. In Section 3 we study labeling problems: we derive a lower bound for a labeling problem that holds also for the case, where *unbounded* preprocessing is allowed; we introduce the t-orientation preprocessing and prove that this preprocessing provides efficient labeling algorithms for certain problems. Section 4 deals with the resource allocation problem: we present the lower bound for k-compact resource allocation, and a distributed algorithm for μ-compact resource allocation using t-orientation. We conclude in Section 5 with some problems, which are left open by our work.

2 Preliminaries

Model of Computation: We consider a distributed message-passing system with n processors $p_1, \ldots, p_n$. The network connecting the processors is modeled as a graph where vertices correspond to processors and there is a bidirectional communication link between every pair of adjacent processors.

We assume the system is synchronous and operates in rounds. That is, at the beginning of round $k+1$, each processor receives all the messages sent to it by its neighbors at the end of round k; after some local computation, the processor may send a message to (some or all of) its neighbors. There is no bound on message length, and local computation is unlimited.

Graph Theoretic Notions: Consider a directed/undirected graph $G = (V, E)$. For any two vertices $v, u \in V$, let $d(u, v)$ be the undirected distance between v and u in G; note that even if G is directed, the distance is measured on the shortest undirected path in G between v and u. The diameter of the G, $diam(G)$, is $\max_{v,u \in V} d(v, u)$. Given a vertex v, the *r-neighborhood* of v, for some integer $r \geq 0$, is the subgraph of G induced by all vertices u such that $d(v, u) \leq r$. The *girth* of G, $g(G)$, is the length of the shortest cycle in G.

A set of vertices $V' \subseteq V$ is an *independent* if no two vertices in V' are adjacent. An independent set is *maximal* if it is not contained in a strictly larger independent set. A *c-coloring* of G is a partition of V into c independent sets. Equivalently, a c-coloring is a mapping $\Psi : V \to \{1, \ldots, c\}$, specifying for each vertex its *color*, such that two adjacent vertices do not have the same color. The *chromatic number* of G, $\chi(G)$, indicates the smallest number c, for which G has a c-coloring.

Given a graph G, denote by $\delta(v)$ the degree of the vertex v, i.e., the number of vertices adjacent to it; let Δ be the maximal degree of a vertex in G.

If G is directed, then a vertex v is a *source* in G if it has no incoming edges.

Labeling Problems: A *labeling* of a graph $G = (V, E)$ with some alphabet Σ is a mapping $\lambda : V \to \Sigma$. A *labeling problem* $\mathcal{L}$ is a set of labelings. Intuitively, this is the set of labelings that satisfy certain requirements. For example, c-coloring is a labeling problem with $\Sigma = \{1, \ldots, c\}$ and the requirement that for every edge $\langle v, u \rangle$, $\lambda(v) \neq \lambda(u)$.

A distributed algorithm solves a labeling problem $\mathcal{L}$ if, after performing some rounds of communication, each processor picks a label such that the labeling of the graph is in $\mathcal{L}$.

3 Labeling Problems

3.1 A Lower Bound

We present a labeling problem and prove that every distributed algorithm for solving this problem requires at least $\Omega(\log n / \log\log n)$ rounds, even with unbounded preprocessing. The problem is a restricted coloring problem, where adjacent vertices should have different labels, and, in addition, the labels have to be close to each other. Formally:

Definition 3.1 *A coloring is* k-dense, *for a fixed* $k \geq 1$, *if every vertex with color* $c > k$ *has a neighbor with color* $c' \in [c - k, c - 1]$.

Intuitively, in a k-dense coloring of a graph, every vertex with color $c > k$ has at least one neighbor with a smaller color which is relatively close to c; k captures the maximal gap between the colors. Given a labeling of a graph, every vertex v with color c can validate its label by examining its 1-neighborhood: the label is legal if v has no neighbor with label c and if $c > k$ then v has a neighbor with color c', $c - k \leq c' \leq c - 1$. (This means that k-dense coloring is 1-checkable, in the terminology of [16].)

We now present our lower bound result. The proof shows a graph G and a vertex $v \in G$, such that v must pick different labels in two different subgraphs G_1 and G_2 of G, but v has the same $\frac{1}{2k}(\log n / \log\log n)$-neighborhoods in G_1 and G_2.

The proof uses graphs which have both a large chromatic number and a large girth; the existence of these graphs is guaranteed by the following theorem.

Theorem 3.1 (Erdös [10]) *For any $n \geq 1$ and ℓ, $4 < \ell < n$, there exists a graph G with n vertices such that $\chi(G) > \frac{1}{2}(\log n)$ and $g(G) > \frac{1}{2}(\log n / \log \ell)$.*

The following is immediate when taking $\ell = \lfloor \log n \rfloor$ in Erdös theorem.

Corollary 3.2 *For any $n \geq 1$ and $2 \leq k < \frac{1}{2}\log n(1 - 1/\log\log n)$, there exists a graph G with n vertices such that $\chi(G) > \frac{1}{2}(\log n / \log\log n) + k$ and $g(G) > \frac{1}{k}(\log n / \log\log n)$.*

The next lemma shows that the maximal color in a k-dense coloring of a tree is a lower bound on the tree's depth.

Lemma 3.3 *In every k-dense coloring of a tree T, if there is a vertex v with color c, then there is a vertex at distance at least $\frac{c}{k} - 1$ ¿from v.*

Proof: Since the coloring is k-dense, v must have a neighbor v_1, such that $c(v_1) \geq c - k$. Similarly, v_1 must have a neighbor v_2, such that $c(v_2) \geq c - 2k$, and in general, v_{i-1} must have a neighbor v_i with color at least $c - ik$. The path $v, v_1, v_2 \ldots v_{i-1}$ can be extended to v_i as long as $i \leq (c-1)/k$. Therefore, the length of the path is at least $\frac{c}{k} - 1$. Clearly, this is a simple path. Since T is a tree, there is no other simple path from v to $v_{\lceil \frac{c}{k} \rceil - 1}$. Therefore, $d(v, v_{\lceil \frac{c}{k} \rceil - 1}) \geq \frac{c}{k} - 1$, which proves the lemma. □

We can now prove the main theorem of this section:

Theorem 3.4 *For every $k > 1$ and $n \geq 1$ such that $k < \frac{1}{2}\log n(1 - 1/\log\log n)$, there exists a communication graph G of size n and a subgraph G' of G such that every distributed algorithm that finds a k-dense coloring of G' requires at least $\frac{1}{2k}(\log n / \log\log n)$ rounds.*

Proof: Assume, by way of contradiction, that there exists an algorithm A which finds a k-dense coloring within R rounds such that $R < \frac{1}{2k}(\log n / \log\log n)$. Clearly, within R rounds a vertex knows only about its R-neighborhood. That is:

Proposition 3.5 *Let G_1 and G_2 be two subgraphs of G, and let v be a vertex of G. If the R-neighborhood of v in G_1 is identical to the R-neighborhood of v in G_2 then v picks the same label when executing A on G_1 and on G_2.*

By Corollary 3.2, there exists a graph G of size n such that $\chi(G) > kR + k$ and $g(G) > 2R$. By the assumption, A finds a k-dense coloring of any subgraph G' of G within R rounds. In particular, A finds a k-dense coloring of G itself. Since $\chi(G) > kR + k$ there exists a vertex v with color $c > kR + k$. Let G' be the R-neighborhood of v in G. Since $g(G) > 2R$ and G' includes only vertices at distance R from v, it follows that G' is a tree. Clearly, v has the same R-neighborhood in G and G'. Therefore, by Proposition 3.5, v is colored c also in G'.

Since G' is a tree, Lemma 3.3 implies that there is vertex at distance $\lceil \frac{c}{k} \rceil - 1$ from v. Hence, $R \geq \frac{c}{k} - 1$. On the other hand, since $c > kR + k$, it follows that $R < \frac{c}{k} - 1$. A contradiction. □

Remark: For stating the lower bound in Theorem 3.4 we assume that $k > 1$. For $k = 1$, the existence of a graph G_1 with n vertices, $\chi(G_1) > \frac{1}{2}(\log n / \log\log n) + 1$, and $g(G_1) > \frac{1}{2}(\log n / \log\log n)$, implies in a similar way, that every distributed algorithm that finds a 1-dense coloring requires at least $\frac{1}{4}(\log n / \log\log n)$ rounds.

3.2 Efficient Labeling Using t-Orientation

In this section we define a class of labeling problems, and show a specific preprocessing which allows to solve them efficiently.

Let $G' \subseteq G$ be a graph that has to be labeled. Clearly, within $diam(G') + 1$ rounds, each processor $v \in G'$ can learn G', and therefore can pick a label.[1] Intuitively, the preprocessing presented in this section orients the edges between neighboring processors, thereby assigning priorities, in such a way that a processor is close to vertices it is oriented to (i.e., with higher priority). We show that for some problems (including coloring and maximal independent set) there exists a labeling algorithm in which a processor's label depends only on the vertices with higher priority. This allows the processor to communicate only with these vertices, which by assumption are relatively close.

3.2.1 t-Orientation of Graphs

We require an acyclic orientation in which every vertex is close to vertices that have a directed path to it.

Definition 3.2 *A* t-orientation *of a graph G is an acyclic orientation (that is, without any directed cycles) of G, such that for every two vertices v and u, if there is a directed path from v to u in the directed graph, then $d(u, v) \leq t$. The* orientation number *of a graph G, denoted by $t(G)$, is the smallest t such that G has a t-orientation.*

Note that for every graph G, topological sorting implies an acyclic orientation, and therefore we have:

Proposition 3.6 *For every graph G, $t(G) \leq diam(G)$.*

However, in most cases we can do much better. For example, any c-coloring of G implies a $(c-1)$-orientation by directing each edge (v, u) from v to u if and only if $color(v) < color(u)$. This is a $(c-1)$-orientation since all directed paths have length at most c. This implies:

[1] Note that if G' is not connected then $diam(G') = \infty$. For some labeling problems, such as coloring, it is sufficient for a processor to know its connected component in G' in order to pick a label. For these problems, the number of rounds needed in order to label G' is $diam(G'_m) + 1$, where G'_m is the connected component with maximal diameter in G'. For other labeling problems, such as finding the number of processors in G', the whole graph G' should be known. For this kind of problems, $diam(G)$ rounds are needed in order to label G'.

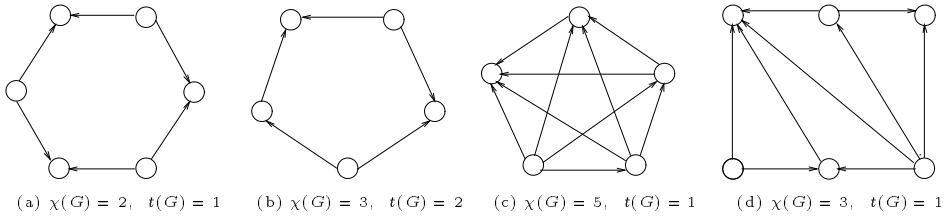

Figure 1: Optimal t-orientations of some graphs.

Proposition 3.7 *For every graph G, $t(G) < \chi(G)$.*

For example, the orientation number of a ring is 1 if the ring is of even length, and 2 if the ring is of odd length (using a 2-coloring or 3-coloring, respectively). Figure 1 includes examples of optimal t-orientations for several graphs.

Recall, that our definition of t-orientation requires only that the *undirected* distance between any two vertices u and v that are connected by a directed path is bounded by t. We comment, that Proposition 3.7 holds also for a stronger definition, that requires the *directed* distance between u and v to be bounded by t. Therefore we expect that the upper bound of $\chi(G)$, as given in Proposition 3.7, can be tightened.[2]

A simple way to construct an optimal t-orientation is by a preprocessing that collects the complete graph topology to some node, and then locally finds the best orientation. (This relies on the fact that local computation power is unbounded.) It requires $O(diam(G))$ communication rounds. In the following we show, that while a moderate computational effort may not yield an optimal orientation, it allows us to find orientations that are good, in the sense that t is always bounded by a small polylogarithm of n.

Theorem 3.8 *For every graph G of size n, it is possible to find an $O((\log n)^2)$-orientation of G by a randomized distributed algorithm within $O((\log n)^2)$ rounds.*

Proof: Every graph can be partitioned into $O(\log n)$ subgraphs $V_1, V_2, \ldots$, such that the diameter of every connected component in these subgraphs is at most $O(\log n)$. This is done by the randomized distributed algorithm of Linial and Saks ([14]) within $O((\log n)^2)$ rounds. At the end of the algorithm, every vertex knows the id, i, of the subgraph V_i to which it belongs, and the ids of the vertices that belong to its connected component in V_i.

This partition can be used to construct an $O((\log n)^2)$-orientation of G within $O(\log n)$ (additional) rounds as follows. Every connected component of every subgraph is oriented acyclicly

[2]The possible gap between $t(G)$ and $\chi(G)$ is well demonstrated in a clique G of n vertices, where $\chi(G) = n$ and $t(G) = 1$.

(e.g., by centralized topological sorting) within $O(\log n)$ rounds. Edges whose endpoints are at different subgraphs are oriented according to the ids of the subgraphs; that is, an edge $\langle v, u \rangle$, with $v \in V_i$ and $u \in V_j$, $i < j$, is oriented $v \rightarrow u$.

Clearly, this orientation is acyclic. Furthermore, assume that there is a directed path from v to u. That path visits the subgraphs defined for G in a strictly increasing order, therefore it visits each subgraph at most once. Since the diameter of every connected component in each subgraph is at most $O(\log n)$, we have $d(v, u) = O((\log n)^2)$. □

3.2.2 Extendible labeling Problems

We now define a class of labeling problems for which the t-orientation preprocessing is helpful. These are problems for which the labeling can be constructed by extending the part of the graph which is already labeled.

Definition 3.3 *Let $G = (V, E)$ be a graph. An* extension *of G is a graph $G' = (V \cup V', E \cup E')$ where $V \cap V' = \emptyset$ and $E' \subseteq V \times V'$. Note that V' is an independent set in G'.*

Definition 3.4 *Let $\mathcal{L}$ be a labeling problem. $\mathcal{A}$ is* an extension labeling algorithm for $\mathcal{L}$ *if for every graph G with a labeling in $\mathcal{L}$, and every extension to $G' = (V \cup V', E \cup E')$, $\mathcal{A}$ gives a label for each $v \in V'$ such that:*

- *The labeling of G' is in $\mathcal{L}$.*
- *For each $v \in V'$, the label of v depends only on the connected components of G to which v is connected. That is, the labeling of v is independent of the labeling of other vertices in V' and of the other components of G.*

Definition 3.5 *A labeling problem is* extendible *if it has a deterministic extension labeling algorithm.*

We now argue that some important labeling problems are extendible. Consider the following extension algorithm for a labeling φ, denoted by A_m:

For every $v \in V'$, $\varphi(v) = 0$ if and only if v has a neighbor $u \in V$ with $\varphi(u) = 1$.

Proposition 3.9 *Finding a maximal independent set is an extendible labeling problem.*

Proof: Let $G = (V, E)$ be a graph which is legally labeled, i.e., every vertex $v \in V$ has a label $\varphi(v) \in \{0, 1\}$ such that the vertices with $\varphi(v) = 1$ form a maximal independent set of G. Let $G' = (V \cup V', E \cup E')$ be an extension of G. A_m is clearly an extension algorithm for the maximal independent set problem. □

Proposition 3.10 *$(\Delta+1)$-coloring is an extendible labeling problem.*

Proof: Let $G=(V,E)$ be a graph which is legally colored, i.e., every vertex $v \in V$ has a label $\psi(v) \in \{1,\ldots,\Delta+1\}$ and for every $c \in \{1,\ldots,\Delta+1\}$, all vertices with $\psi(v)=c$ form an independent set. Let $G'=(V \cup V', E \cup E')$ be an extension of G. The following is clearly an extension algorithm for this problem:

For every $v \in V$, define $\psi(v)$ to be the smallest $c \in \{1,\ldots,\delta(v)+1\}$ such that no neighbor u of v has $\psi(u)=c$. Such c exists because v has $\delta(v)$ neighbors, and therefore at most $\delta(v)$ colors are used by v's neighbors. For each $v \in V'$, $\delta(v) \leq \Delta$, thus G' is $(\Delta+1)$-colored. □

An extension algorithm that labels a vertex with the smallest color not used by its neighbors is suitable for the k-dense coloring problem. Therefore:

Proposition 3.11 *For every $k \geq 1$, the k-dense coloring problem is extendible.*

3.2.3 An Algorithm for Extendible Labeling Problems

Theorem 3.12 *Given a t-orientation of a graph G, for any extendible labeling problem $\mathcal{L}$, there is a distributed algorithm that solves $\mathcal{L}$ within t rounds on every subgraph of G.*

Proof: Let $\mathcal{L}$ be an extendible labeling problem, and let $\mathcal{A}$ be a deterministic extension algorithm for $\mathcal{L}$. We describe a distributed algorithm that solves $\mathcal{L}$ on any subgraph of G within t rounds.

Let G be a graph with an acyclic orientation, and let G' be a subgraph of G. Note that the t-orientation of G induces an acyclic orientation of G'. Consider a partition of G' into layers $L_0(G'), L_1(G'), \ldots, L_{max}(G')$, where *max* is the length of the longest directed path in G'. For any $v \in G'$, $v \in L_i(G')$ if and only if the longest directed path to v in G' is of length i. Note that this partition is well defined since G is finite and the orientation is acyclic.

Claim 3.13 *Each layer $L_i(G')$ forms an independent set.*

Proof: Let v and u be neighbors in G', such that $v \rightarrow u$. Every directed path to v can be extended to u, and in particular the longest one. Thus, u belongs to a layer higher than v's layer. □

For every vertex $v \in G'$, let $G'_{in}(v)$ be the subgraph of G' induced by v and all the vertices in G' that have a directed path to v. For each $v \in G'$, we partition $G'_{in}(v)$ into the layers $L_0(G'_{in}(v)), L_1(G'_{in}(v)), \ldots, L_k(G'_{in}(v))$, where k is the length of the longest directed path in $G'_{in}(v)$. This partition has the following properties:

$i \leftarrow 0$
Already-labeled $\leftarrow \emptyset$
repeat
    Execute $\mathcal{A}$(Already-labeled , $L_i(G'_{in}(v))$)
    Already-labeled $\leftarrow$ Already-labeled $\cup L_i(G'_{in}(v))$
    $i \leftarrow i + 1$
until v is labeled.

Figure 2: The labeling algorithm: code for $v \in G'$

- If $u \in G'_{in}(v)$ then every directed path to u in G' is in $G'_{in}(v)$; that is, $G'_{in}(u) \subseteq G'_{in}(v)$.
- In particular, if $u \in G'_{in}(v)$, then the longest directed path to u is in $G'_{in}(v)$; therefore, for every i and v, $L_i(G'_{in}(v)) \subseteq L_i(G')$.
- Consequently, if $u \in G'_{in}(v)$ then for every i, $L_i(G'_{in}(u)) \subseteq L_i(G'_{in}(v))$.

The algorithm consists of two stages. In the first stage, information is collected. Specifically, during the first t rounds, every vertex $v \in G'$ distributes to distance t the fact that it belongs to G'. All the vertices of G participate in this stage. Since G is t-oriented, each vertex $v \in G'$ knows $G'_{in}(v)$ within t rounds.

In the second stage of the algorithm, every vertex $v \in G'$ uses $\mathcal{A}$, the extension algorithm, to label $G'_{in}(v)$. The labeling is computed in iterations. In the ith iteration, v labels $L_i(G'_{in}(v))$. The code for $v \in G'$ for this stage appears in Figure 2.

We denote by $\mathcal{A}(H, V)$ the application of $\mathcal{A}$ when the labeled graph $H \subseteq G'_{in}(v)$ is extended by an independent set V and all the edges which connect H and V in G'. On each iteration of the **repeat** loop, an additional layer of $G'_{in}(v)$ is labeled. Denote by $label_v(u)$ the label assigned by v to $u \in G'_{in}(v)$, when v executes $\mathcal{A}$. In particular, $label_v(v)$ is the label that v assigns to itself.

The next lemma shows that the labels v assigns to vertices in $G'_{in}(v)$ are identical to the labels those vertices assign to themselves.

Lemma 3.14 *If $u \in G'_{in}(v)$, then $label_u(u) = label_v(u)$.*

Proof: We show, by induction on $i \geq 0$, that $label_u(u) = label_v(u)$, for every $u \in (G'_{in}(v) \cap L_i(G'))$.

The base case is $i = 0$. Note that $L_0(G')$ contains the sources of G'. Consider some $u \in L_0(G')$ and note that $label_u(u)$ is determined in the first iteration, when u executes $\mathcal{A}(\emptyset, u)$. Every v such that $u \in G'_{in}(v)$ assigns a label to u in the first iteration by executing

$\mathcal{A}(\emptyset, L_0(G'_{in}(v)))$. There may be some other vertices in addition to u in $L_0(G'_{in}(v))$, but since the label of u depends only on its connected component which includes only u, and since $\mathcal{A}$ is deterministic, $label_u(u) = label_v(u)$.

For the induction step, assume that the claim holds for all vertices in $L_i(G')$ for $i < j$. Consider $u \in L_j(G')$, and note that $label_u(u)$ is determined in the jth iteration, when u executes $\mathcal{A}(\bigcup_{i<j} L_i(G'_{in}(u)), u)$. Every v such that $u \in G'_{in}(v)$ assigns $label_v(u)$ when it executes $\mathcal{A}(\bigcup_{i<j} L_i(G'_{in}(v)), L_j(G'_{in}(v)))$. The connected component of u in $\bigcup_{i<j} L_i(G'_{in}(v))$ is $\bigcup_{i<j} L_i(G'_{in}(u))$. By the induction assumption, all the vertices of both $\bigcup_{i<j} L_i(G'_{in}(u))$ and $\bigcup_{i<j} L_i(G'_{in}(v))$ are labeled identically by v and by u. Thus, since $\mathcal{A}$ is deterministic, $label_u(u) = label_v(u)$. □

The entire labeling of G' consists of the labels $label_v(v)$. By Lemma 3.14, it is identical to the labeling produced by $\mathcal{A}$ when applied to G' sequentially, layer by layer. Thus, it is in $\mathcal{L}$. □

By Theorem 3.8, we have:

Corollary 3.15 *For every graph G of size n, after a randomized preprocessing that takes $O((\log n)^2)$ rounds, any extendible labeling problem can be solved on every $G' \subseteq G$ within $O((\log n)^2)$ rounds.*

Note that for the preprocessing suggested in the above results we assume that n is known in advance.

Proposition 3.11 and Theorem 3.12 imply that for every graph G and a fixed $k \geq 1$, a k-dense coloring of every $G' \subseteq G$ can be found distributively within t rounds, assuming the existence of a t-orientation of G. In particular, by Corollary 3.15, there is a randomized distributed preprocessing that takes $O((\log n)^2)$ rounds, and enables to find a k-dense coloring of every $G' \subseteq G$, within $O((\log n)^2)$ rounds. Note that the lower bound for k-dense coloring, from Theorem 3.4, is $\Omega(\log n / \log\log n)$.

Since the k-dense coloring problem is extendible, Theorem 3.4 and Theorem 3.12 imply:

Corollary 3.16 *Let $t(n)$ be the maximal $t(G)$ among graphs of size n.*
Then $t(n) = \Omega(\log n / \log\log n)$.

4 Resource Allocation

In this section we study the resource allocation problem. This problem, in contrast to labeling problems, has an "on-going" nature and has to be repetitively solved for each instance.

However, as will be shown below, we employ techniques and results that were developed for labeling problems.

An instance of the resource allocation problem is a *communication graph* G, where the vertices represent processors, and there is an edge between any pair of processors that may compete on some resource. The resource requirements of a processor may vary. The current requirements are represented formally in a dynamic *conflict graph* C, where the vertices are processors waiting to execute their jobs, and there is an edge between two processors that compete on some resource. Clearly, $C \subseteq G$. We denote the degree of processor p_i in the conflict graph C by δ_i, and by μ be the maximum number of rounds required to complete a job.

An algorithm for the resource allocation problem decides when each waiting processor can use the resources and execute its job; it should satisfy to following properties:

1. *Exclusion:* No two conflicting jobs are executed simultaneously. (That is a safety property.)
2. *No starvation:* The request of any processor is eventually granted. (This is a liveness property.)

The *response time* for a request is the number of rounds that elapse ¿from the processor's request to use resources until it executes the job. A good algorithm should minimizes the response time. We consider also the following property, that guarantees better exploitation of the resources, and reduces the average response time:

Definition 4.1 *An algorithm for the resource allocation is* k-compact *for every waiting processor* p_i*, if in every* k *rounds either* p_i *runs or some conflicting neighbor of* p_i *runs.*

In Section 4.1 we prove that for every $k \geq 1$ there is no efficient distributed algorithm which is k-compact, by reduction to the lower bound for k-dense coloring, that was proved earlier. Specifically, we show a lower bound of $\Omega(\lg n / \lg\lg n)$ on the response time of any resource allocation algorithm that is k-compact, for any $k \geq 1$. Section 4.2 presents the compact coloring problem, which is used later for compact resource allocation. In Section 4.3 we present a distributed μ-compact algorithm for resource allocation, which uses the t-orientation preprocessing.

4.1 A Lower Bound for k-Compact Resource Allocation

We show that given a conflict graph C and $k \geq 1$, any k-compact resource allocation algorithm can be used to label C such that the labeling is a $\lceil \frac{k}{\mu} \rceil$-dense coloring. Together with the lower bound proved in Theorem 3.4 this implies the lower bound for compact resource allocation.

Let G be a communication graph and let C be a conflict graph. The *one-shot resource allocation problem* is to schedule the resources for C in a way that satisfies the safety and liveness conditions. A *slow* execution for a given set of jobs is an execution where each job uses the resources for exactly μ rounds. (This terminology is borrowed from Rhee [19].)

For a specific algorithm, consider a slow execution with respect to the one-shot resource allocation problem. That is, the algorithm has to schedule only one "batch" of jobs, each of which needs the resources for the same running time, μ. Clearly, this is a special case of the resource allocation problem and any lower bound for this case applies to the general problem.

Let t_0 be the first round in which some processor starts executing its job; the no starvation property guarantees the existence of t_0. Associate with each processor p_i a label $\lambda(p_i)$ such that $\lambda(p_i) = c$ if and only if p_i starts executing its job in the interval $[t_0 + (c-1)\mu\,,\, t_0 + c\mu)$. Such an interval exists by the no starvation property, and hence the labeling is well defined.

Claim 4.1 *The labeling λ is a $(\lceil \frac{k}{\mu} \rceil + 2)$-dense coloring of the conflict graph.*

Proof: By the mutual exclusion property, and since the execution is slow, λ is a legal coloring.

Assume now that $\lambda(p_i) = c > \lceil \frac{k}{\mu} \rceil + 2$. That is, p_i starts executing its job in the interval $[t_0 + (c-1)\mu\,,\, t_0 + c\mu)$. Since the algorithm is k-compact, in every k rounds, either p_i starts executing its job, or there exists some conflicting processor p_j which executes its job. In the latter case, there is a conflicting processor, p_j, which starts executing its job in the interval $[t_0 + (c-2)\mu - k\,,\, t_0 + (c-1)\mu)$. By the definition of λ, p_j is labeled c', $(c-2) - \frac{k}{\mu} \leq c' \leq c-1$, as needed. □

Together with Theorem 3.4, this implies:

Theorem 4.2 *For every $k \geq 1$, there is no k-compact distributed algorithm for the resource allocation problem with response time less than $\frac{\mu}{2k+6\mu}(\log\, n / \log\log\, n)$.*

4.2 Compact Coloring

In this section we introduce the compact coloring problem and its properties. In the next section, we use these properties to show that processors joining the conflict graph C at different times in our algorithm, agree on the same colors for processors in C.

Definition 4.2 *A coloring is* compact *if every vertex v with color j has neighbors with all colors $1, \ldots, j-1$.*

Note that every compact coloring is 1-dense. On the other hand, consider a graph that is a line of length 4, whose vertices are colored $1, 2, 3, 4$. This is a 1-dense coloring which is not compact.

$V_1 = V$
$E_1 = E$
$i \leftarrow 1$
Repeat
    Execute A_m on $G_i = (V_i, E_i)$
    $\mathrm{MIS}_i \leftarrow \mathrm{MIS}(G_i)$ produced by A_m
    For every $v \in \mathrm{MIS}_i$, $\Phi(v) \leftarrow i$
    $V_{i+1} \leftarrow V_i \setminus \mathrm{MIS}_i$
    $E_{i+1} \leftarrow E_i \setminus \{(u,v) \mid u, v \in \mathrm{MIS}_i\}$
    $i \leftarrow i + 1$
Until G_i is empty

Figure 3: Algorithm A_m^*.

For a given compact coloring, let C_i denote the set of vertices colored with i; since the coloring is compact, C_i is a maximal independent set in $V \setminus \bigcup_{j<i} C_j$. Consider the following extension algorithm for a labeling Ψ, denoted by A_c: For every $v \in V$, define $\Psi(v)$ to be the smallest number c such that no neighbor u of v has $\Psi(u) = c$. Clearly:

Lemma 4.3 *Compact coloring is an extendible labeling problem.*

The next lemma claims that if we remove all the vertices colored 1 by A_c ¿from a graph G, then we obtain a graph G' such that for every vertex v in G', if v was colored c in G then v is colored $c-1$ when applying A_c to G'.

Lemma 4.4 *Let $G = (V, E)$ be a graph, and let $\Psi : V \to N$ be the compact coloring of G, produced by A_c. Let $G' = (V', E')$ be the graph obtained by deleting all the vertices for which $\Psi(v) = 1$ from G, and let $\Psi' : V' \to N$ be the compact coloring of G' produced by A_c. Then for every $v \in V'$, $\Psi'(v) = \Psi(v) - 1$.*

To prove the lemma, we consider the algorithm A_m^* which iteratively executes the MIS extension algorithm, A_m, on a given graph G (see Figure 3). Recall, that A_m labels a vertex v with 1 if v has no neighbor from a lower layer which is labeled 1; otherwise, v is labeled with 0. A_m^* is useful for studying A_c, due to the following claim:

Claim 4.5 *For every graph G, the labeling Φ produced by A_m^* is identical to the labeling Ψ produced by A_c.*

Proof: Recall, that given an acyclically oriented graph G, a vertex v is in the ith layer, $L_i(G)$, if and only if the longest directed path to v is of length i. The proof is by induction on the layers of G.

For the base case, consider a vertex $v \in L_0(G)$. Note that $L_0(G)$ contains the sources of G. Both A_c and A_m^* color every source v with 1.

For the induction step, assume that the claim holds for all vertices in $L_j(G), j < i$, and let $v \in L_i(G)$. Assume that $\Phi(v) = k$, that is, $v \in \mathrm{MIS}_k$. Consider the neighbors of v from lower layers at the end of iteration k of A_m^* in which v joins MIS_k. By A_c, v is colored k if and only if v has neighbors from lower layers with all colors $\{1, \ldots, k-1\}$, and no neighbor from lower layers which is colored k.

Since every iteration of A_m^* produces a maximal independent set, v has neighbors from lower layers in $\mathrm{MIS}_1, \mathrm{MIS}_2, \ldots, \mathrm{MIS}_{k-1}$. By the induction hypothesis, this implies that v has neighbors which are colored $1, \ldots, k-1$ by A_c. By A_m, v joins MIS_k if and only if v has no neighbor from lower layers in MIS_k. Thus, by the induction hypothesis, v has no neighbor from a lower layer which is colored k by A_c. Therefore, $\Psi(v) = k$, as needed. □

Proof of Lemma 4.4: By Claim 4.5, $\Phi(v) = \Psi(v)$, for every $v \in G$. In particular, $\mathrm{MIS}_1(G)$ is the set of vertices with $\Psi(v) = 1$. Remove $\mathrm{MIS}_1(G)$ from G. By Claim 4.5, the resulting graph is G'. Let Φ' be the coloring produced by applying A_m^* to G'.

Consider the execution of A_m^* on G. By A_m^*, $\mathrm{MIS}_1(G)$ is removed from G after the first iteration of that execution. Since the resulting graph is G', the remainder of this execution on G is identical to the execution of A_m^* on G'. That is, the execution of A_m^* on G' is identical to the suffix of the execution on G starting from the second iteration. Hence, $v \in \mathrm{MIS}_i(G)$ if and only if $v \in \mathrm{MIS}_{i-1}(G')$. This implies that for every $v \in V'$, $\Phi'(v) = \Phi(v) - 1$. By Claim 4.5, for every $v \in V'$, $\Psi'(v) = \Psi(v) - 1$. □

By repeatedly removing the set of vertices which are colored 1, we obtain:

Corollary 4.6 *Let $G = (V, E)$ be a graph, and let $\Psi : V \to N$ be the compact coloring of G, produced by the extension algorithm A_c. For a fixed integer $z \geq 0$, let $G' = (V', E')$ be the graph obtained by deleting all the vertices for which $\Psi(v) \in \{1, \ldots, z\}$ ¿from G, and let $\Psi' : V' \to N$ be the compact coloring of G' produced by A_c. Then for every $v \in V'$, $\Psi'(v) = \Psi(v) - z$.*

This corollary is used in our resource allocation algorithm to show that processors joining the conflict graph C at different times, agree on the same colors for processors in C.

4.3 A Distributed μ-Compact Resource Allocation Algorithm

In this section we describe a μ-compact distributed algorithm for the resource allocation problem, whose response time is $\delta_i \mu + 2(t(G) + 1)$, where $t(G)$ is the orientation number of the communication graph G.

We assume that μ is known in advance and processors can fix *running phases*, each consisting of μ rounds. In addition, processors submit their requests for resources in *entrance phases*, each consisting of $t(G)+1$ rounds. A processor wishing to execute a job waits for the beginning of the next entrance phase and then submits its request. This adds at most $t(G)+1$ rounds to the response time of every request. The partitions of rounds to entrance phases and running phases are identical with respect to all the processors. Therefore, processors submit requests in batches, with $t(G)+1$ rounds between two successive batches.

The algorithm uses a preprocessing which finds an acyclic orientation of G which achieves the orientation number of G; we use $p_i \rightarrow p_j$ to denote that p_i is oriented to p_j. The orientation and the entrance phases induce an orientation of the dynamic conflict graph C as follows: An edge $\langle p_i, p_j \rangle$ is directed $p_i \Rightarrow p_j$ if p_i requests resources in an earlier entrance phase than p_j or if p_j and p_i request resources in the same entrance phase and $p_i \rightarrow p_j$.

For each entrance phase, the processors are partitioned into three sets:

1. *Idle:* Processors that do not need resources, and processors that are currently executing their jobs.
2. *Requesting:* Processors that request resources in the current entrance phase.
3. *Waiting:* Processors that requested resources in previous entrance phases and are still waiting for their running phase.

The idea of the algorithm is to use the t-orientation in order to merge the requesting processors with the waiting processors, in a manner that does not delay the waiting processors and provides short response time for the new requests. The code for processor p_i appears in Figure 4. As in Section 3.2, we denote by $C_{in}(p_i)$ the subgraph of C such that $p_j \in C_{in}(p_i)$ if and only if there is a directed path $p_j \Rightarrow \ldots \Rightarrow p_i$ in C.

Intuitively, the algorithm proceeds as follows. Each requesting processor p_i transmits its requests and collects the current state of $C_{in}(p_i)$. Upon having the initial state of $C_{in}(p_i)$, denoted by $C^0_{in}(p_i)$, the running phase of p_i is determined by a compact coloring of $C^0_{in}(p_i)$. If p_i is colored k then p_i executes its job in the kth running phase, counting from the first running phase that begins after the end of the current entrance phase. The waiting processors update the conflict graph and transmit it to the requesting processors. At the beginning of each entrance phase the updated state of $C_{in}(p_i)$ is obtained from the previous one by deleting the set of processors that will begin executing their jobs in the next $t(G)+1$ rounds.

First, we prove that every requesting processor p_i learns about processors that may influence its color during its entrance phase.

Lemma 4.7 *A requesting processor, p_i, knows $C^0_{in}(p_i)$ at most $t(G)+1$ rounds after the beginning of its entrance phase.*

Do every entrance phase:

If you do not need resources:

In the next $t(G)+1$ rounds:

Transmit to your neighbors all the messages you receive.

In order to execute a job:

In the next round:

Receive from your neighbors the part of $C_{in}(p_i)$ which consists of processors who made requests in previous entrance phases.

In the next $t(G)$ rounds:

Distribute that part of $C_{in}(p_i)$ and your request

Transmit to your neighbors all the messages you receive.

Construct $C_{in}(p_i)$ by combining the old part you already know with the parts you received in the last $t(G)$ rounds.

Use A_c to find a compact coloring of $C_{in}(p_i)$.

If you are colored k then execute your job in the kth running phase.

For every $p_j \in C_{in}(p_i)$,

If p_j is colored k then p_j executes its job in the kth running phase.

If you are waiting:

Update $C_{in}(p_i)$:

Remove processors that will start executing their job in the next $t(G)+1$ rounds according to your compact coloring.

Remove processors which are not connected to you anymore.

Distribute $C_{in}(p_i)$ to your neighbors.

In the next $t(G)$ rounds:

Transmit to your neighbors all the messages you receive.

Figure 4: The distributed algorithm: code for p_i

Proof: The proof is by induction on the entrance phase. For the base case, consider a processor p_i that requests resources in the first entrance phase. Directed paths to p_i contain only other processors that request resources in the first entrance phase. Since G was t-oriented, the distance between each processor in $C_{in}(p_i)$ and p_i is at most $t(G)$. Therefore, p_i knows $C_{in}(p_i)$ after at most $t(G)$ rounds.

For the induction step, let p_i be a processor that requests resources in the rth entrance phase, $r > 1$. Let $\rho = p_j \Rightarrow \ldots \Rightarrow p_i$ be a directed path to p_i in C. By the algorithm, no processor that enters with p_i is directed to a processor from an earlier entrance phase. Thus, ρ can be divided into two parts $p_j \Rightarrow \ldots \Rightarrow p_k \Rightarrow p_l \Rightarrow \ldots \Rightarrow p_i$ such that $p_j, \ldots, p_k$ request resources strictly before the rth entrance phase, and $p_l, \ldots, p_i$ request resources in the rth entrance phase.

Two successive entrance phases are separated by $t(G)+1$ rounds. Therefore, by the inductive hypothesis, when p_i joins, p_k already knows the path $p_j \Rightarrow \ldots \Rightarrow p_k$. By the algorithm, p_l receives from p_k this part of ρ in the first round of the rth entrance phase. Since the graph is t-oriented, p_i receives messages from all the vertices in $p_l \Rightarrow \ldots \Rightarrow p_i$ within $t(G)$ rounds and can reconstruct ρ. □

The next lemma states that for every $p_j \in C^0_{in}(p_i)$, the assignments of a running phase to p_j as done by p_i and p_j are identical. That is, p_j is colored k in the compact coloring of $C^0_{in}(p_i)$ if and only if p_j is going to execute its job in the kth running phase, counting from the first running phase that begins after the end of p_i's entrance phase.

Lemma 4.8 *For every requesting processor p_i and for every $p_j \in C^0_{in}(p_i)$, the running phase assigned to p_j by p_i is k if and only if p_j executes its job in phase k.*

Proof: The proof is by induction on the entrance phase. For the base case, consider a processor p_i that requests resources in the first entrance phase. Since p_j is in $C^0_{in}(p_i)$, p_j also submits requests in the first entrance phase. By Lemma 4.7, p_i knows $C^0_{in}(p_i)$ within $t(G)+1$ rounds. From Lemma 4.3, the compact coloring problem is extendible. Therefore, by Claim 3.14 (that refers to the Algorithm A_c), p_j assigns color k to itself if and only if p_i assigns color k to p_j in $C^0_{in}(p_i)$. Since p_i and p_j start counting from the same round, the running phases are counted identically by p_i and p_j. This implies the lemma.

For the induction step, assume that the induction hypothesis holds for all processors that request resources before the rth entrance phase, and let p_i be a processor that submits requests at the rth entrance phase. By the algorithm, at the first round of phase r, every waiting processor p_l removes from $C_{in}(p_l)$ processors that will start executing their jobs during entrance phase r, according to the compact coloring calculated by p_l. $C_{in}(p_l)$ includes only processors that request resources before the rth entrance phase. Thus, by the induction hypothesis, these updates reflect correctly the current state of $C_{in}(p_l)$. This fact, together with Lemma 4.7, implies that p_i obtains $C^0_{in}(p_i)$ within $t(G)+1$ rounds. Consider a processor $p_j \in C^0_{in}(p_i)$. There are two cases:

Case 1: p_j is a processor that requests resources in entrance phase r. By the definition of $C_{in}(p)$, $C^0_{in}(p_j) \subseteq C^0_{in}(p_i)$, and using Claim 3.14, p_j assigns color k to itself if and only if p_i assigns color k to p_j in $C^0_{in}(p_i)$.

Case 2: p_j is a processor that requests resources in entrance phase r'. Note that $r > r'$, since $C^0_{in}(p_i)$ does not include any processors that request resources after p_i. Let x be the ratio between the length of one entrance phase and the length of one running phase, that is, $t(G) + 1 = x \cdot \mu$. Let s denote the number of the first running phase to begin after the rth entrance phase, and let s' denote the number of the first running phase to begin after the r'th entrance phase That is, $s' = s - \lfloor x(r - r') \rfloor$. Assume that p_j assigns to itself color c' at entrance phase r'. By the algorithm, p_j will execute its job at running phase $s' + c'$. In addition, during the $r - r'$ entrance phases between r' and r, p_j removes from $C_{in}(p_j)$ all the processors that will start executing their jobs during that period. Formally, all the processors colored by p_j with $1, \ldots, \lfloor x(r - r') \rfloor$ are removed from $C_{in}(p_j)$. The induction hypothesis implies that the updated $C_{in}(p_j)$ at the beginning of the rth entrance phase contains only processors which are still waiting.

By Corollary 4.6, a compact coloring of the updated $C_{in}(p_j)$ assigns color $c = c' - \lfloor x(r - r') \rfloor$ to p_j. Since $C_{in}(p_j) \subseteq C^0_{in}(p_i)$, Claim 3.14 implies that p_i assigns color c to p_j in $C^0_{in}(p_i)$. Thus, p_i determines that p_j executes its job in running phase number $s + c = s' + \lfloor x(r - r') \rfloor + c = s' + c'$, and that completes the proof. □

We can now prove the main properties of the algorithm.

Lemma 4.9 (Safety) *For every two processors p_i and p_j, if $need_i \cap need_j \neq \emptyset$ then p_i and p_j do not run simultaneously.*

Proof: If $need_i \cap need_j \neq \emptyset$ then p_i and p_j are neighbors in C. Assume, without loss of generality, that $p_i \Rightarrow p_j$ and hence $p_j \in C_{in}(p_i)$. Since $C_{in}(p_i)$ is legally colored, p_i and p_j have different colors in $C_{in}(p_i)$. By Lemma 4.8, the running phase that p_i determines for p_j is identical to the running phase that p_j determines for itself. Therefore, p_i and p_j belong to different running phases. Thus, by the algorithm, p_i and p_j do not run simultaneously. □

We now show that the schedule becomes μ-compact at most $2(t(G) + 1)$ rounds after a processor initiates a request for resources.

Lemma 4.10 *For every waiting processor p_i, after the first $2(t(G) + 1)$ rounds, in every μ rounds either p_i runs or some conflicting neighbor of p_i runs.*

Proof: By the algorithm, for every waiting processor p_i, the coloring of $C_{in}(p_i)$ is compact. Thus, if p_i is colored c, it has neighbors with all colors $1, \ldots, c - 1$. Therefore, there is at least one neighbor of p_i which runs in each of the running phases $1, \ldots, c - 1$. The first running phase in this count of the running phases begins at most $2(t(G) + 1)$ rounds after the request was initiated by p_i. Hence, after that round the schedule is μ-compact. □

The response time for a processor p_i consists of three components: First, p_i waits for the beginning of the next entrance phase, which takes at most $t(G)+1$ rounds. Then, during the entrance phase, p_i collects $C^0_{in}(p_i)$. By Lemma 4.7, this takes $t(G)+1$ rounds. Finally, p_i waits for its running phase. By the μ-compact property (Lemma 4.10), p_i waits at most δ_i running phases, each taking μ rounds. This implies:

Theorem 4.11 *There exists an algorithm for the resource allocation problem whose response time is* $\delta_i\mu + 2(t(G)+1)$.

Remark: In our algorithm, δ_i captures the number of processors that issued competing resource requests before or simultaneously with p_i. That is, a processor is not delayed because of processors that request resources after it. Note, that in general it does not mean that the algorithm guarantees a FIFO ordering. Thus, a processor p_j that issued its request later than p_i may execute its job earlier (while p_i is still waiting). This happens only if p_i needs a "popular" resource that was not requested by p_j.

4.4 Discussion

As presented, the algorithm assumes that the system is synchronous, and that the local computing power at the processors is unlimited.

First, we remark that the algorithm can be easily changed to work in asynchronous systems, by employing a simple synchronizer, such as α [1]. Since our algorithm rely on synchronization only between neighboring processors, synchronizer α allows to run the algorithm correctly. The details, which are straightforward, are omitted.

Second, we remark that the local computation performed in our algorithm is fairly moderate. The most consuming step is the computation of a compact coloring; this is done by repeated application of A_m, which in turn, greedily assigns colors to nodes. Furthermore, this computation can be integrated with the collection of information from neighboring nodes. This way, the compact coloring is computed in iterations that overlap the iterations in which information is collected; the local computation at each node reduces to choosing a color, based on the colors of its neighbors.

5 Conclusions and Open Problems

This work addressed the power of unrestricted preprocessing, in particular, the t-orientation preprocessing. Several open questions remain:

1. We derive a lower bound on the number of communication rounds needed for a k-compact resource allocation. Is there a lower bound on the number of communication rounds needed for a resource allocation algorithm that guarantees only the safety and liveness properties?

2. Our lower bound for k-dense coloring depends on k, while our upper bound for this problem is the same for all values of k. Can these bounds be tightened? In particular, is there an algorithm for k-dense coloring whose complexity depends on k?

3. We show that the t-orientation preprocessing helps in some labeling problems. Are there other helpful types of preprocessing?

4. We show that $t(G) \leq O((\log n)^2)$ for every graph G of size n, and that there exists a graph G of size n such that $t(G) = \Omega(\log n / \log\log n)$. Can the upper bound be reduced to $O(\log n / \log\log n)$? In particular, is there a distributed algorithm that achieves a better orientation? Is there a non-randomized distributed algorithm that achieves a good orientation?

5. Is it NP-Hard to determine $t(G)$ for a given graph?

Acknowledgments: We would like to thank Roy Meshulam for bringing Erdös' Theorem to our attention and for pointing out the existence of graphs with $t(G) = \Omega(\log n / \log\log n)$. We also thank Amotz Bar-Noy for helpful discussions. An anonymous referee provided many comments that improved the presentation.

References

[1] B. Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM,* Vol. 32, No. 4 (1985), pp. 804–823.

[2] B. Awerbuch and D. Peleg. "Sparse partitions," *IEEE Symp. on Foundation of Computer Science,* 1990, pp. 503–513.

[3] B. Awerbuch and M. Saks. "A dining philosophers algorithm with polynomial response time." *IEEE Symp. on Foundation of Computer Science,* 1990, pp. 65–74.

[4] V.C. Barbosa and E. Gafni. "Concurrency in heavily loaded neighborhood-constrained systems." *ACM Trans. Programming Languages and Systems,* Vol. 11, 1989, pp. 562–584.

[5] J. Bar-Ilan and D. Peleg. "Distributed resource allocation algorithms." *International Workshop on Distributed Algorithms,* 1992, pp. 276–291.

[6] A. Bar-Noy, H. Shachnai and T. Tamir. "On Chromatic Sums and Distributed Resource Allocation." *4th. Israel Symposium on Theory and Computing Systems,* 1996.

[7] K. Chandy and J. Misra. "The drinking philosophers problem." *ACM Trans. Programming Languages and Systems,* Vol. 6, 1984, pp. 632–646.

[8] M. Choy and A. K. Singh. "Efficient fault tolerant algorithms in distributed systems." *Proc. 24th ACM Symposium on Theory of Computing,* 1992, pp. 593–602.

[9] R. Cole and U. Vishkin. "Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms." *Proc. 18th ACM Symposium on Theory of Computing,* 1986, pp. 206–219.

[10] P. Erdös. "Graph theory and probability." *Canad.J.Math., Vol.11*, 1959, pp. 34–38.

[11] A. Goldberg, S. Plotkin and G. Shannon. "Parallel symmetry-breaking in sparse graphs." *Proc. 19th ACM Symposium on Theory of Computing,*, 1987, pp. 315–324.

[12] N. Linial. "Distributive algorithms—Global solutions from local data." *IEEE Symp. on Foundation of Computer Science,* 1987, pp. 331–335.

[13] N. Linial. *Local-global phenomena in graphs.* Technical Report #9, Hebrew University, Jerusalem, Israel, 1993.

[14] N. Linial and M. Saks. "Decomposing graphs into regions of small diameter." *SIAM-ACM Symposium on Discrete Algorithms,* 1991, pp. 320–330.

[15] A. Mayer, M. Naor and L. Stockmeyer. "Local computations on static and dynamic graphs." *3rd. Israel Symposium on Theory and Computing Systems,* 1995.

[16] M. Naor and L. Stockmeyer. "What can be computed locally?" *25th ACM Symposium on Theory of Computing,* 1993, pp. 184–193.

[17] A. Panconesi and A. Srinivasan. "Improved distributed algorithms for coloring and network decomposition problems." *24th ACM Symposium on Theory of Computing,* 1992, pp. 581–592.

[18] M. Szegendy and S. Vishwanathan. "Locality based graph coloring." *25th ACM Symposium on Theory of Computing,* 1993, pp. 201–207.

[19] I. Rhee. *Efficiency of partial synchrony, and resource allocation in distributed systems.* Ph.D Thesis, University of North Carolina at Chapel Hill, June 1994.