Non-clairvoyant Scheduling with Conflicts for Unit-Size Jobs

Hongyang Sun[a]

[a]*Department of EECS, Vanderbilt University, Nashville TN, USA*

Abstract

We consider the problem of scheduling unit-size jobs with conflicts in a non-clairvoyant setting. In this problem, all jobs are released at the same time but the conflict graph is unknown to the scheduling algorithm beforehand. Instead, the conflicts between the jobs are revealed online by an adversary during the jobs' execution. There is an unlimited number of processors so that all jobs could be executed simultaneously. However, two jobs with a conflict cannot be both completed in the same time step; when their conflict is revealed, one of them must be aborted and re-executed later. The objective is to minimize the total number of time steps required to complete all jobs, or the makespan.

We present an online non-clairvoyant algorithm and analyze its performance for three types of conflict graphs. For bipartite graphs, we show that it is $O(\log n)$-competitive, where n denotes the total number of jobs. For unit-interval graphs, we show that it is $O(\log \chi)$-competitive, where χ denotes the chromatic number of the graph. For bounded-degree graphs, we show that it is $O(\Delta)$-competitive, where Δ denotes the maximum degree of any node in the graph. For bipartite and bounded-degree graphs, we also provide matching lower bounds and show that the obtained competitive ratios are asymptotically tight.

Keywords: Competitive analysis, online algorithms, conflict graphs, non-clairvoyant scheduling, makespan minimization

1. Introduction

We consider the problem of scheduling jobs with conflicts in a non-clairvoyant setting, where the conflicts between the jobs are unknown to the scheduling algorithm beforehand. We call the problem *Non-clairvoyant Scheduling with Conflicts (NSC)*. The NSC problem has practical applications in the contention management of transactional memory systems, where a set of transactions execute concurrently with potential conflicts between them (e.g., when two transactions attempt to write to the same memory location). The conflicts are typically unknown to the contention manager/scheduler, which needs to decide on-the-fly which transaction to abort when a conflict is detected and when to restart the aborted transaction [8, 1]. In this paper, we focus on transactions that have equal conflict-free execution time, or equivalently jobs of unit size.

1.1. Scheduling Model

Suppose a set of n unit-size jobs are all released at the same time and need to be scheduled on m processors. Time is divided into discrete steps of unit size, so each job, without conflict with other jobs, can be executed by one processor and thus completes in one time step. To focus on the conflict-resolving aspect of the scheduling problem, we assume that the number of processors is unlimited or is at least the number of jobs (i.e., $m \geq n$), so that all jobs can be executed simultaneously at any time step. However, two jobs with a conflict between them cannot be both completed in the same time step. The conflicts of the jobs are modeled by a *conflict graph* $G = (V, E)$, where each node $v \in V$ represents a job and an undirected edge $e = (u, v) \in E$ represents a conflict between two jobs v and u. The objective is to minimize the *makespan*, i.e., the total number of time steps required to complete all jobs. Note that when the conflict graph is known to the scheduling algorithm a priori, the problem is offline and is equivalent to the graph-coloring problem. In this case, the chromatic number χ of the conflict graph represents the optimal makespan.

In this paper, we study an *online non-clairvoyant* version of the problem, where the conflict graph is unknown to the scheduling algorithm beforehand, and it is revealed progressively by an adversary. In particular, an online

Email address: `hongyang.sun@vanderbilt.edu` (Hongyang Sun) algorithm starts out with zero knowledge of the conflict graph, and gains more knowledge about it during the course of execution. Hence, it is possible that two jobs, say u and v, with a conflict between them are scheduled to execute together in a time step, because the online algorithm is not aware of this conflict. During such a time step, the conflict will be revealed by an adversary (the adversarial model will be described below), and the algorithm must respond immediately by aborting one job, say u, and the other job v can then continue its execution until it is either completed or aborted (due to a subsequently revealed conflict between v and another job, say w) within the same time step. In the latter case (i.e., v also aborted), the conflict (u, v) will become known to the online algorithm, and this knowledge could help the algorithm make better scheduling decisions in the later time steps. In general, an online algorithm at any time step must make the following two decisions.

- *Job selection decision*: Select a subset of the pending jobs to execute for each time step. Here, a job is said to be *pending* if it is not yet completed prior to the time step.
- *Job aborting decision*: Decide which job will be aborted and which job will continue to execute when a conflict between two jobs is revealed during the execution of a time step.

For simplicity, we only allow jobs to start executing at the beginning of a time step, not arbitrarily in the middle, thus preventing the execution of a job from spanning two time steps. Moreover, we can assume without loss of generality that the subset of pending jobs selected by an online algorithm for each time step does not contain conflicts that are already learnt by the algorithm (it may contain unknown conflicts). That is, the selected jobs form an independent set based on the algorithm's current knowledge of the conflict graph. Otherwise, the adversary can always force the algorithm to resolve these known conflicts before revealing any new one.

1.2. Adversarial Model

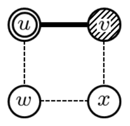
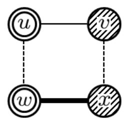
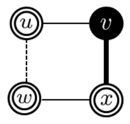
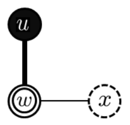
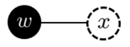
What makes the NSC problem difficult is the power possessed by the adversary, which can limit the information gained by the online algorithm. The only requirement for the adversary is that it should reveal conflicts in such a way that prevents two conflicting jobs from being completed at the same time step. However, the adversary has certain flexibility or power in the conflict-revealing process. In particular, we assume the following adversarial model:

- The adversary can reveal a conflict at any time during a time step (e.g., in the middle or towards the end), so an aborted job cannot be rescheduled until the next time step.
- The adversary can decide the order in which multiple conflicts are revealed during each time step, possibly adapting its decisions based on the online algorithm's response;
- The adversary can hide certain conflicts during a time step if it becomes unnecessary to reveal them (e.g., when the incident jobs are not selected or already aborted/completed).

To illustrate the adversarial model with an example, consider a conflict graph $G = (V, E)$, where $V = \{u, v, w, x\}$ and $E = \{(u, v), (u, w), (w, x), (x, v)\}$. Table 1 illustrates a possible conflict-revealing sequence by the adversary along with an online algorithm's job selection and aborting decisions during each time step of the execution. Note that time step 1 consists of three sub-steps where three conflicts are revealed in sequence: (u, v), (w, x), (x, v), and the online algorithm is forced to make an aborting decision immediately after each conflict is revealed. Observe that the conflict (u, w) is hidden by the adversary during this first time step, since both of its incident jobs u and w have already been aborted by the algorithm following the previously revealed conflicts. This allows the adversary to reveal (u, w) in the second time step, which forces the algorithm to respond with another abort.

We point out that the conflict graph, although initially unknown to the online algorithm, is fixed a priori and cannot be changed by the adversary after the algorithm has (partially) discovered it. Hence, a better algorithm for the above example should abort job u instead of w in time step 2, and execute both jobs u and x in time step 3. This will result in a makespan of 3 instead of 4. It is not hard to see that any deterministic online non-clairvoyant algorithm for this example can be forced to make the "wrong" decision either in job selection or in job aborting. Thus, only one job could be completed in the first time step and it would take at least two more steps to complete the remaining three jobs which contain two conflicts. This implies that the online algorithm cannot produce a makespan smaller than 3 under the considered adversarial model while the optimal offline makespan is obviously 2.

Table 1: The execution of a conflict graph $G = (V, E)$, where $V = \{u, v, w, x\}$ and $E = \{(u, v), (u, w), (w, x), (x, v)\}$. In the illustration figures, solid lines (—) represent edges in the conflict graph that are already learnt by the algorithm, thick lines (▬) represent edges (or conflicts) that are revealed in each time step or sub-step, and dashed lines (……) represent the edges (or conflicts) that are still hidden from the algorithm; white nodes (○) represent jobs that are currently running, double-circled nodes (◎) represent jobs that are aborted after a conflict is revealed, shaded nodes (▨) represent jobs that continue to execute after an aborting decision, black nodes (●) represent jobs that are completed in the time step, and dashed nodes (◌) represent jobs that are not selected for the time step.

Time step	Job selection (by algorithm)	Conflict revealed (by adversary)	Job aborting (by algorithm)	Illustration
1	$\{u, v, w, x\}$	(u, v)	abort u, continue v	
		(w, x)	abort w, continue x	
		(x, v)	abort x, complete v	
2	$\{u, w\}$	(u, w)	abort w, complete u	
3	$\{w\}$	*nil*	complete w	
4	$\{x\}$	*nil*	complete x	

1.3. Main Results

Despite much related work on online graph coloring and conflict scheduling (see Section 2 for a literature review), the NCS problem, to the best of our knowledge, has not been explicitly studied. In this paper, we study the NCS problem while focusing on three specific types of conflict graphs, namely bipartite graphs, unit-interval graphs and bounded-degree graphs. The performance of an online non-clairvoyant algorithm is measured using *competitive analysis*. An online algorithm is said to be *c-competitive* if its makespan is at most c times the optimal offline makespan, which is the chromatic number of the conflict graph.

The following summarizes the main contributions of this paper:

- We present an online non-clairvoyant scheduling algorithm for the NSC problem that is generally applicable to any conflict graph.
- For bipartite graphs, we prove that the algorithm is $O(\log n)$-competitive, where n is the total number of jobs. We also prove a matching lower bound for any deterministic online non-clairvoyant algorithm.
- For unit-interval graphs, we prove that the algorithm is $O(\log \chi)$-competitive, where χ is the chromatic number of the graph (or the optimal offline makespan). The same ratio also applies to some other unit geometric intersection graphs, such as unit-disk graphs and unit-square graphs.

- For bounded-degree graphs, we prove that the algorithm is $O(\Delta)$-competitive, where Δ is the maximum degree of any node in the graph. We also show that the competitive ratio is tight up to a factor of two for any deterministic online non-clairvoyant algorithm.

The rest of this paper is organized as follows. Section 2 reviews some related work on online graph coloring and conflict scheduling. Sections 3 presents our proposed non-clairvoyant scheduling algorithm, followed by its analysis for the three special conflict graphs in Section 4. Finally, Section 5 concludes the paper with some remarks and future directions.

2. Related Work

The offline version of the NSC problem is exactly the graph-coloring problem, which is known to be NP-hard and no polynomial-time algorithm can approximate the chromatic number within $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$ [6]. The best algorithm to date has an approximation ratio of $O(n(\log\log n)^2/\log^3 n)$ [9]. For the case where there is a constant number m of machines and the number n of jobs is more than m, the problem can be formulated as an m-set-cover problem [5] by first constructing independent sets of size at most m. Applying the greedy algorithm for the set cover problem then gives H_m-approximation, where H_m denotes the m-th harmonic number. Moreover, an exact algorithm exists for scheduling on $m = 2$ machines even when the job sizes are in $\{1, 2\}$ [5]. The related problem of minimizing the total (weighted) completion time of the jobs has also been studied in [3, 4, 7].

In an online version of the graph coloring problem, jobs or vertices are revealed one by one by the adversary, and once a job is presented, its conflicts with the previously presented jobs are also revealed. A color must be irrevocably assigned to each revealed job respecting the existing conflicts, and the objective is to minimize the total number of colors used. For this problem, the competitive ratio of any online algorithm has been shown to be $\Omega(n/\log^2 n)$ [10] and the best algorithm so far achieves a competitive ratio of $O(n/\log^* n)$ [14]. However, algorithms with better performance ratios exist for some special graphs. For example, bipartite graphs admit an $O(\log n)$-competitive online algorithm [14], a 3-competitive algorithm exists for interval graphs [13], and the First-Fit (FF) algorithm is $O(\log n)$-competitive for inductive graphs [11].

In a related online problem, jobs have release times and a released job reveals all its conflicts with the existing ones. Two conflicting jobs cannot be scheduled together, and the objective is to minimize the makespan. A lower bound of 2 on the competitive ratio of any deterministic/randomized algorithm was shown in [15], and a 2-competitive algorithm was proposed for unit-size jobs and a 3-competitive algorithm for jobs with arbitrary sizes [15]. Both algorithms are not restricted computationally. The competitive ratio for the latter case was later improved to 2 [5].

In yet another related problem motivated by traffic signal control [12], the conflict graph is given in advance, and the jobs arrive online at different nodes. Only one job from each node can be scheduled at any time, but two jobs from a pair of conflicting nodes cannot be scheduled together. The objective is to minimize the maximum response time of any job that enters the system. It was shown that any deterministic online algorithm is $\Omega(n)$-competitive for bipartite graphs and interval graphs [12], where n is the total number of nodes. Online algorithms were also presented whose competitive ratios depend on both n and the maximum response time in the optimal schedule.

The NSC problem considered in this paper is primarily motivated by the management of contentions in transactional memory systems. Many related works exist in this context (see, e.g., [8, 2, 1, 17, 18]). However, most of these works are based on the over-pessimistic assumption that the conflict graph can be altered freely by the adversary after a transaction is aborted, which often leads to unreasonably large lower bounds (in the order of the number of jobs or the number of resources that cause the contentions). One work [16] that does not make this assumption proposed a randomized algorithm and proved that it achieves a competitive ratio of $O(\max\{\Delta, \log n\})$ with high probability for graphs with maximum degree Δ. In this paper, we present an $O(\Delta)$-competitive algorithm for bounded-degree graphs when the algorithm is not restricted computationally. To the best of our knowledge, no previous work has explicitly studied the NSC problem for general graphs and for the other special graphs considered in this paper.

3. Algorithm

We present an online non-clairvoyant algorithm that is generally applicable to unit-size jobs in any conflict graph for the NSC problem. We will show the performance of the algorithm for three special graphs in the next

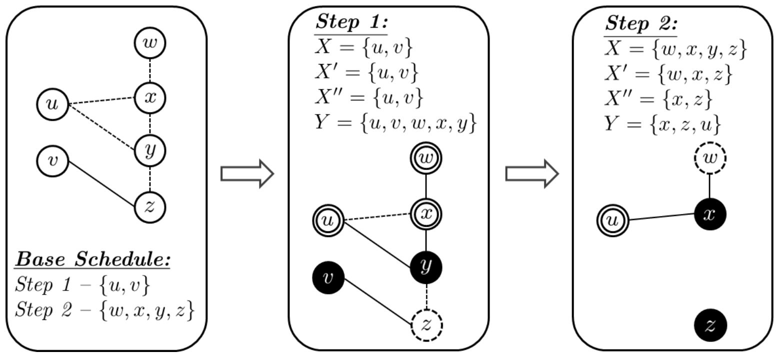


Figure 1: Illustration of the job selection policy for a partially known conflict graph in a round, where only one conflict (v, z) is known. The base schedule of the round consists of two time steps. In the first time step, three more jobs w, x, y are added to the initial set $X = \{u, x\}$, forming the final set $Y = \{u, v, w, x, y\}$ of selected jobs. During this time step, the conflicts $(u, y), (w, x), (x, y)$ are revealed, and jobs v, y are completed while jobs u, w, x are aborted. In the second time step, job y is first removed from the initial set $X = \{w, x, y, z\}$ since it has completed in the previous step. Job w is then removed from the remaining set $X' = \{w, x, z\}$ due to its conflict with job x, which has become known. Lastly, job u is added to the remaining set $X'' = \{x, z\}$ because it has no known conflict with jobs x, z, forming the final set $Y = \{x, z, u\}$ of selected jobs. During this time step, the conflict (u, x) is revealed, and jobs x, z are completed while job u is aborted. In the figures within the two time steps, solid lines (—) represent conflicts that become known to the algorithm and dashed lines (......) represent conflicts that are still hidden from the algorithm at the end of the step; double-circled nodes (◎) represent jobs that are aborted, black nodes (●) represent jobs that are completed, and dashed nodes (◌) represent jobs that are not selected for the time step.

section. This section describes its two scheduling decisions and show some general properties.

3.1. Job Selection

In our algorithm, jobs are scheduled in *rounds*. A round consists of one or more time steps, which may be different from round to round. At the beginning of each round, the algorithm computes an optimal schedule for all pending jobs based on the existing knowledge of the conflict graph.[1] We call this schedule the *base schedule*. The following describes the job selection policy for each time step within a round.

Job selection policy: At the beginning of each time step,

(a) select the set of jobs that are scheduled to run at this time step according to the base schedule. Denote this set by X.

(b) remove from X the jobs that are already completed prior to this time step (these jobs may exist due to (d) below). Denote the remaining set by X'.

(c) if X' contains jobs with known conflicts (again possible due to (d) below), remove from X' any job in each pair of conflicted jobs. Denote the remaining set by X''.

(d) consider each remaining job outside set X that is still pending and add the job to X'' if it does not have a known conflict with any job in X''. Denote the final set by Y, which is selected to execute in this time step.

Figure 1 illustrates the job selection policy by showing the base schedule of a partially known conflict graph for a round together with the sets of jobs (X, X', X'', Y) considered in each of its two time steps. Note that the final set Y of selected jobs in a time step may differ from the set X originally planned in the base schedule because of the possible job removals and additions in the process. A round ends when all the jobs have completed or when all the time steps in the planned base schedule have elapsed. In the latter case, if there are still pending jobs, a new round will be planned.

The following shows some simple properties of the proposed job selection policy.

[1]Finding an optimal schedule may be NP-hard for some conflict graphs (even when the graph is only partially known). As in some previous studies of online models [15, 5], we assume that the algorithm is not restricted computationally. The goal is to bound the performance of an online algorithm due to lack of knowledge rather than limitation in computing power.

Property 1. *(1) The number of time steps in each round is at most the chromatic number χ of the conflict graph; (2) Any pending job is executed at least once in a round; (3) At any time step, a pending job either executes itself or has an immediate neighbor in the conflict graph that executes.*

Proof. (1) Each round starts with an optimal schedule of a subgraph. Thus, its length is at most the chromatic number of the complete conflict graph. Moreover, subsequent job removals and additions do not increase the number of time steps; (2) A pending job is executed either at its designated time step in the base schedule, or at an earlier time step due to (d) in the policy; (3) Again, due to (d) in the policy, the final set of jobs selected at any time step forms a maximal independent set based on the current knowledge of the conflict graph. Thus, if a pending job is not selected, an immediate neighbor must be selected. □

3.2. Job Aborting

Before describing the job aborting decision, we first introduce the notion of *aborting trees*, which are constructed by tracking the aborting relationships of the running jobs during their executions. Specifically, at the beginning of each time step, the selected jobs form their own aborting trees, each containing a single node (i.e., the job itself). These trees will be merged when jobs begin to be aborted. For each job $v \in V$, we define s_v to be the *score* associated with the job, which at any time indicates the total number of nodes in its rooted subtree (including the job itself). Initially, all selected jobs have a score of 1. The following describes the job aborting policy.

> ***Job aborting policy***: When a conflict between two jobs u and v is revealed, abort the job with the lower score, say job u. The other job v inherits the aborting tree rooted at u as its subtree, as well as the score of u, i.e., $s_v^{new} = s_v^{old} + s_u$. Ties are broken arbitrarily if two jobs have the same score.

According to the policy above, at the end of each time step, the root node of each aborting tree is completed and all of its descendants remain pending. These aborting trees will be abandoned at the end of the time step and new aborting trees will be constructed for the next time step. The following property shows a lower bound on the total number of nodes in an aborting tree with a given height.

Property 2. *An aborting tree with height h has at least 2^{h-1} nodes.*

Proof. We prove the property by induction on the height of the aborting tree. For the base case of $h = 1$, there is only 1 job in the tree, and the claim holds trivially. Suppose the claim holds for any $h \geq 1$. To show that it also holds for a tree with height $h+1$, consider the first abort that makes the aborting tree reach height $h+1$. Let v and u denote the surviving job and the aborted job respectively in this case. Clearly, the aborted job u must be the root of a tree with height h, and from the inductive hypothesis, its score is at least $s_u \geq 2^{h-1}$. According to the policy, the score of the surviving job v before aborting u must be at least that of u, i.e., $s_v^{old} \geq s_u$. Hence, after aborting u and inheriting its score, the score of job v, or the number of nodes in its rooted subtree, satisfies $s_v^{new} = s_v^{old} + s_u \geq 2 \cdot 2^{h-1} = 2^h$. □

4. Analysis

In this section, we prove the competitive ratios of the presented algorithm for three special conflict graphs, namely, bipartite graphs, unit-interval graphs, and bounded-degree graphs. We also show that the ratios for bipartite graphs and bounded-degree graphs are asymptotically tight.

4.1. Bipartite Graphs

For any bipartite graph $G = (U, V, E)$, its vertices can be divided into two disjoint sets (or parts) U and V such that every edge $e \in E$ connects a vertex in U to a vertex in V. In other words, each part of a bipartite graph contains an independent set of jobs/vertices[2] without any conflict among them.

Clearly, any bipartite graph has a chromatic number of two and hence each execution round in the algorithm consists of at most two time steps. Before each round, the set of all jobs, including those completed and the pending ones, form a collection of *connected components* based on the known conflicts among them. A connected

[2]We use the terms *job* and *vertex* interchangeably.

component is said to be *pending* if it contains at least one pending job. Moreover, if the conflict graph is bipartite, an optimal base schedule can be computed efficiently by performing a graph traversal and assigning any initial node of a pending connected component to an arbitrary part and each subsequent node to the opposite part as its parent node. The following lemma shows that the number of pending connected components reduces at least exponentially over rounds.

Lemma 1. *After each round, the number of pending connected component is reduced by at least half.*

Proof. There is no internal conflict among jobs within each part of a connected component, otherwise an odd cycle will appear contradicting the fact that the supergraph is bipartite. Thus, any conflict revealed must come from two jobs from two pending connected components. Regardless of the job aborting policy, the two connected components will be merged via the newly discovered edge. Hence, a pending connected component is either completed without encountering any conflict after a round or merged with at least one more connected component. □

Based on the result of Lemma 1, the following theorem shows the competitive ratio of the algorithm.

Theorem 1. *Our algorithm is $O(\log n)$-competitive for jobs with bipartite conflict graphs.*

Proof. Let c_i denote the number of pending connected components after the i-th round, and let $c_0 = n$. Lemma 1 suggests that the number of pending connected components after round i satisfies $c_i \leq \lfloor n/2^i \rfloor$, so only one pending connected component will remain after at most $\lg n$ rounds. Since each round has at most two time steps, this implies that our algorithm completes all jobs in at most $2(\lg n + 1)$ time steps. □

While Theorem 1 shows that our algorithm achieves $O(\log n)$-competitive on bipartite graphs, the following theorem proves that this bound is asymptotically tight for any deterministic algorithm.

Theorem 2. *Any deterministic online non-clairvoyant algorithm is $\Omega(\log n)$-competitive on bipartite graphs.*

Proof. We first prove the lower bound by assuming that the number of jobs is a power-of-2, i.e., $n = 2^k$ for some $k \in \mathbb{Z}^+$. The assumption will be relaxed later for arbitrary n. The bipartite graph is given by a rooted tree defined iteratively as follows: Let T_0 be a single node, and for each $i = 1, \cdots, k$, construct tree T_i by connecting the root nodes of two copies of T_{i-1} and designating any one of them as the new root. Before each time step, the adversary ensures that the online algorithm does not possess any knowledge about the conflicts of the pending jobs. Thus, we assume that the algorithm is greedy, i.e., it will execute all pending jobs in a time step (again, this assumption will be relaxed later). In particular, the adversary at time step i only reveals the conflicts involving the leaf nodes of tree T_{k-i+1}. Since an online algorithm does not know the hidden conflicts and hence the overall tree structure, the aborted jobs can be forced by the adversary to be the parent nodes and only the leaf nodes will be completed, leaving tree T_{k-i} to the next step $i + 1$. Figure 2 illustrates the adversarial strategy for $k = 3$ and $n = 8$ with the jobs' executions in each step. Clearly, any deterministic algorithm will take $k + 1 = \lg n + 1$ time steps to complete all jobs, while the optimal makespan is 2.

We now relax the two assumptions made above. First, if n is not power-of-2, we can construct a rooted tree using $n' = 2^{\lfloor \lg n \rfloor}$ nodes and make the remaining $n - n'$ nodes independent (i.e., without any conflict with any other node). Under the same adversarial strategy, any greedy algorithm that execute all pending jobs in a time step will produce a makespan of $\lg n' + 1 = \lfloor \lg n \rfloor + 1 > \lg n$. Second, for any deterministic algorithm that is non-greedy and only executes a subset of all pending jobs, the adversary can force it to select the leaf (or independent) nodes before the internal nodes. This may possibly reduce the number of conflicts the adversary needs to reveal at each step, thus making the algorithm perform worse than the greedy strategy. □

4.2. Unit-Interval Graphs

For any interval graph $G = (V, E)$, its vertex set $V = \{I_1, I_2, \cdots, I_n\}$ represents a set of intervals on the real line, and an edge $e \in E$ connects a pair of vertices if the corresponding intervals intersect each other. An interval graph is called unit-interval graph if all of its intervals have unit length. A unit-interval graph is also known as a proper interval graph.

Clearly, the vertices in any interval graph with chromatic number χ can be partitioned into χ disjoint sets, and each set contains non-intersecting intervals. We show that our algorithm completes all jobs in a unit-interval conflict graph in $O(\chi \log \chi)$ time steps without knowing the chromatic number χ.

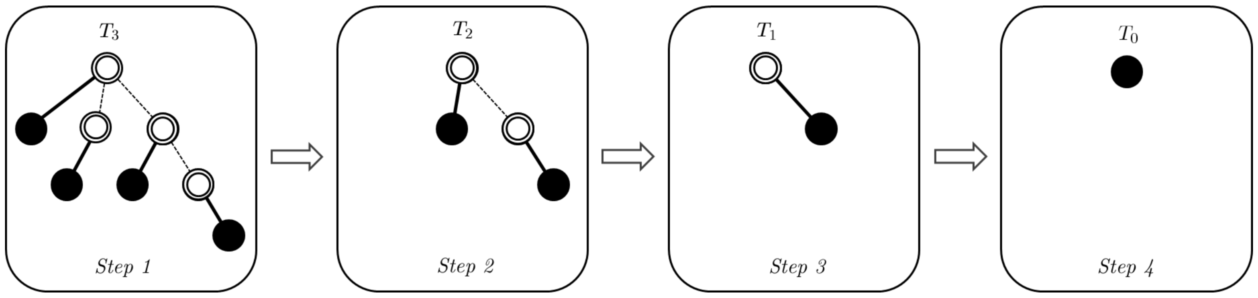


Figure 2: The adversarial strategy for $k = 3$ and $n = 8$ in bipartite graphs. The thick solid lines (▬) represent the conflicts revealed by the adversary in each time step and the dashed lines (......) represent the hidden conflicts. Any deterministic online non-clairvoyant algorithm is forced to abort the double-circled nodes (◎) during each step while the black nodes (●) are completed.

We say that a job I_i is within *q-neighborhood* of another job I_j if there exists a path from I_i to I_j in the conflict graph that contains at most q edges. For a unit-interval graph, the following lemma gives an upper bound on the number of jobs within q-neighborhood of a given job.

Lemma 2. *For any $q \in \mathbb{Z}^+$ and any unit-interval graph with chromatic number χ, there are at most $(2q+1)\chi$ jobs that are within q-neighborhood of a given job.*

Proof. For any given job I in the graph, let ℓ_I denote the left endpoint of the corresponding interval I on the real line. Observe that any unit interval that is within q-neighborhood of I must be completely contained in the larger interval $\mathcal{I} = [\ell_I - q, \ell_I + 1 + q]$. Since the chromatic number of the graph is χ, one can pack at most $(2q+1)\chi$ unit intervals in $\mathcal{I}$ without introducing a clique of $\chi + 1$ intervals. ☐

With the help of Lemma 2 and Properties 1 and 2, the following lemma bounds the height of any aborting tree and the distance from a pending job to a completed job in a time step.

Lemma 3. *Any pending job is within $O(\log \chi)$-neighborhood of a completed job in a time step.*

Proof. Let h denote the height of any aborting tree. According to Property 2, the total number N of jobs in the aborting tree is at least $N \geq 2^{h-1}$. Since all jobs in the aborting tree are within $(h-1)$-neighborhood of the root of the tree, Lemma 2 suggests that the total number of jobs in the tree is at most $N \leq (2(h-1)+1)\,\chi = (2h-1)\chi$. Thus, for the inequality $2^{h-1} \leq N \leq (2h-1)\chi$ to hold for all $\chi \geq 2$, the height of the tree must satisfy $h = O(\log \chi)$. According to Property 1.3, any pending job either executes itself in the time step thus is in an aborting tree, or is within 1-neighborhood of another executed job that is in an aborting tree. In both cases, the job is within $h = O(\log \chi)$-neighborhood of the root of an aborting tree, which is completed in the same time step. ☐

The following theorem proves the competitive ratio of our algorithm.

Theorem 3. *Our algorithm is $O(\log \chi)$-competitive for jobs with unit-interval conflict graphs, where χ denotes the chromatic number of the graph.*

Proof. Let $I \in V$ denote the last completed job in the job set. Based on Lemma 2, there are at most $O(\chi \log \chi)$ jobs that are within $O(\log \chi)$-neighborhood of I. According to Lemma 3, job I in the worst case may need to wait for all of these jobs to complete before I itself can be completed, which implies a makespan of $O(\chi \log \chi)$ time steps, while the optimal makespan is obviously χ. ☐

Remarks. The bound shown in Theorem 3 also holds for some other unit geometric intersection graphs, such as unit-disk graphs and unit-square graphs. In these graphs with two-dimensional geometric objectives, there are at most $O(q^2\chi)$ jobs that are within q-neighborhood of a given job, assuming that the chromatic number of the graph is χ. The analysis in Lemma 3 nevertheless gives the bound of $O(\log \chi)$, which leads to the same asymptotic competitive ratio.

4.3. Bounded-Degree Graphs

Finally, we consider conflict graphs with bounded maximum degree Δ. In this case, a randomized algorithm was presented in [16], which achieves a competitive ratio of $O(\max\{\Delta, \log n\})$ with high probability. We show that our algorithm, which is unrestricted computationally, is $O(\Delta)$-competitive.

Theorem 4. *Our algorithm is $O(\Delta)$-competitive for jobs with bounded-degree conflict graphs, where Δ denotes the maximum degree of any node in the graph.*

Proof. According to Property 1.2, any pending job is executed at least once in a round. Thus, at the end of the round, the job either completes or discovers at least one new conflict with another job. This implies that there exist at most $\Delta + 1$ rounds, since a job can conflict with at most Δ other jobs. Also, Property 1.1 shows that each round consists of at most χ time steps, where χ is the optimal makespan. Thus, the makespan of our algorithm is bounded by $(\Delta + 1)\chi$. $\square$

Remarks. The competitive ratio of the algorithm is tight up to a factor of 2. To see that, consider the instance shown in the proof of Theorem 2, where a bipartite graph with $n = 2^k$ nodes has maximum degree $\Delta = k$. Any deterministic algorithm in this case is shown to produce a makespan at least $k + 1 = \Delta + 1$ while the optimal makespan is 2.

5. Concluding Remarks

In this paper, we have studied the NSC problem to minimize the makespan for a set of unit-size jobs, and have presented an online non-clairvoyant algorithm with proven competitive ratios for three special conflict graphs. The problem for other special graphs or general graphs remains open. One interesting direction is to improve the competitive ratio for unit-interval graphs or to provide a matching lower bound. Other possible directions include considering jobs with arbitrary sizes, optimizing alternative objective functions (e.g., total response time), and restricting the number of processors.

Acknowledgement

I would like to thank Rui Fan for his inspiration and many helpful discussions. I also thank the anonymous reviewers for valuable suggestions that help to improve the presentation of this paper.

References

[1] H. Attiya, L. Epstein, H. Shachnai and T. Tamir. Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica*, 57(1):44–61, 2010.

[2] H. Attiya and A. Milani. Transactional scheduling for read-dominated workloads. In *OPODIS*, pages 3–17, 2009.

[3] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[4] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman and H. Shachnai. Sum multicoloring of graphs. *Journal of Algorithms*, 37(2):422–450, 2000.

[5] G. Even, M. M. Halldórsson, L. Kaplan and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

[6] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer Systems and Sciences*, 57(2):187–199, 1998.

[7] R. Gandhi, M. M. Halldórsson, G. Kortsarz and H. Shachnai. Improved bounds for scheduling conflicting jobs with minsum criteria. *ACM Transactions on Algorithms*, 4(1):1549–6325, 2008.

[8] R. Guerraoui, M. Herlihy and B. Pochon. Toward a theory of transactional contention managers. In *PODC*, pages 258–264, 2005.

[9] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.

[10] M. M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130(1):163–174, 1994.

[11] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.

[12] S. Irani and V. Leung. Scheduling with conflicts on bipartite and interval graphs. *Journal of Scheduling*, 6(3):287–307, 2003.

[13] H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143-153, 1981.

[14] L. Lovász, M. Saks and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1-3):319–325, 1989.

[15] R. Motwani, S. Phillips and E. Torng Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[16] J. Schneider and R. Wattenhofer. Bounds on contention management algorithms. *Theoretical Computer Science*, 412(32):4151–4160, 2011.

[17] G. Sharma and C. Busch. A competitive analysis for balanced transactional memory workloads. *Algorithmica*, 63(1-2):296–322, 2012.

[18] G. Sharma, B. Estrade, and C. Busch. Window-based greedy contention management for transactional memory: theory and practice. *Distributed Computing*, 25(3):225–248, 2012.