Asynchronous Coordination Under Preferences and Constraints*

Armando Castañeda[1], Pierre Fraigniaud[2], Eli Gafni[3], Sergio Rajsbaum[1], and Matthieu Roy[4]

[1] Instituto de Matemáticas, UNAM, México D.F, 04510, México,
armando.castaneda@im.unam.mx, rajsbaum@im.unam.mx
[2] CNRS and University Paris Diderot, France,
pierref@liafa.univ-paris-diderot.fr
[3] Computer Science Department, UCLA, USA, eli@cs.ucla.edu
[4] LAAS-CNRS, Universit de Toulouse, CNRS, Toulouse, France. roy@laas.fr

Abstract. *Adaptive renaming* can be viewed as a *coordination task* involving a set of asynchronous agents, each aiming at grabbing a single resource out of a set of resources Similarly, *musical chairs* is also defined as a coordination task involving a set of asynchronous agents, each aiming at picking one of a set of available resources, where every agent comes with an a priori *preference* for some resource. We foresee instances in which some combinations of resources are allowed, while others are disallowed. We model these *constraints* as an undirected graph whose nodes represent the resources, and an edge between two resources indicates that these two resources cannot be used simultaneously. In other words, the sets of resources that are allowed are those which form *independent sets*.

We assume that each agent comes with an a priori *preference* for some resource. If an agent's preference is not in conflict with the preferences of the other agents, then this preference can be grabbed by the agent. Otherwise, the agents must coordinate to resolve their conflicts, and potentially choose non preferred resources. We investigate the following problem: given a graph, what is the maximum number of agents that can be accommodated subject to non-altruistic behaviors of early arriving agents?

Just for cyclic constraints, the problem is surprisingly difficult. Indeed, we show that, intriguingly, the natural algorithm inspired from optimal solutions to adaptive renaming or musical chairs is sub-optimal for cycles, but proven to be at most 1 to the optimal. The main message of this paper is that finding optimal solutions to the *coordination with constraints and preferences* task requires to design "dynamic" algorithms, that is, algorithms of a completely different nature than the "static" algorithms used for, e.g., renaming.

* The 1st and 4th authors are supported by UNAM-PAPIIT IA101015 and IN107714. The 1st author is also supported by the project CONACYT C394/2016/271602. The 4th author also received support from ECOS-CONACYT and LAISLA. The 2nd author received support from the ANR project DISPLEXITY, and from the INRIA project GANG. The 5th author is supported by CPSLab project H2020-ICT-644400.

1 Introduction

1.1 Context and objective

In distributed computing, several tasks have their *adaptive* versions in which the quality of the solution must depend only on the number of processes that participate in a given execution, and not on the total number of processes that could be involved in this task. A typical example of an adaptive task is *adaptive renaming* [4]. In renaming, each process is aiming at acquiring a *name* taken from a small range of integers $[1, r]$, under the constraint that all acquired names must be pairwise distinct. The *quality* of a renaming algorithm is judged based on the range r of names, the smaller the better. In adaptive renaming, r must depend only on the number k of participating processes. In the *asynchronous* setting with *crash-prone processes* and *read/write registers*, the optimal value for the range is known to be $r = 2k - 1$ [5,13].

Interestingly, adaptive renaming can also be viewed as a *task* by interpreting the integers $1, \ldots, r$ as a total order on the names, where name i is preferred to name j whenever $i < j$. Hence, adaptive renaming can be viewed as an abstraction of the problem in which asynchronous *agents* are competing for *resources* totally ordered by their desirability. In other words, adaptive renaming is an abstraction of a problem of *coordination* between agents under *preferences*. Coordination between agents under preferences has been recently investigated in [2,3] where the *musical chairs* game has been formally defined and solved. In this game, a set of players (modeling the agents) must coordinate so that each player eventually picks one of the available chairs (modeling the resources). Each player initially comes with an a priori *preference* for one chair. In absence of conflict with other players, the player can pick the desired chair, otherwise the conflicting players must coordinate so that they pick different chairs. It is proved that the smallest number r of chairs for which musical chairs with k players has a solution is $r = 2k - 1$.

We foresee that neither adaptive renaming nor musical chairs fully capture typical scenarios of agents competing for resources. Indeed, both tasks only capture scenarios in which the constraint is that no two agents can acquire the same resource. In practice, resources may not be independent, and the literature on scheduling, partitioning, resource allocation, etc. (see, e.g., [6,7,11,15,16]) provide several examples of problems in which resources are inter-dependent, causing some resource a not being allowed to be used simultaneously with resource b. That is, using one resource disables others. In this paper, we consider the case in which constraints are modeled as an indirected graph whose nodes are resources, and every edge $\{a, b\}$ indicates that resources a and b cannot be both simultaneously acquired, i.e., acquiring a node disables all its neighbors. In other words, the sets of resources that are allowed are those which form *independent sets* in the graphs. In this framework, renaming as well as musical chairs correspond to the case where the graph of constraints is a stable one (i.e., a graph with no edges). We thus address an extension of renaming and musical chairs, targeting an abstraction of a problem of coordination between agents under *constraints*.

Our objective is to understand the power and limitation of coordination between agents competing for interdependent resources. We are focussing on a scenario inspired from musical chairs in which a resource is a priori assigned to each agent, and the agents have to coordinate between them so that to eventually acquire pairwise non conflicting resources. In particular, if the initial assignment forms an independent set, then the agents do not have to do anything. Alternatively, if they are initially assigned conflicting resources, then they have to spread out and coordinate themselves so that they eventually acquire a set of resources that form an independent set. In other words, each agent comes with an a priori *preference* for some resource — these preferences for the resources do not need to be different. If an agent's preference is not in conflict with the preference of another agent, then it can grab its preference. Otherwise, this agent must choose another resource.

The coordination task between agents under preferences and constraints is thus defined as follows. Given an n-node graph $G = (V, E)$ modeling the constraints between the resources, an input is a multiset M of k elements in V representing the preferences of k processes $p_1, \ldots, p_k$ modeling the k agents. Outputs are independent sets $I = \{u_1, \ldots, u_k\}$ in G, of size k representing the fact that process p_i acquires u_i, for $i = 1, \ldots, k$. The literature on renaming [5] and musical chair [3] taught us that, in an asynchronous system in which the processes are subject to crash failures, the task is not solvable for k larger than some bound, even for the stable graph G (the value of the bound on k for the stable graph is roughly half the number of nodes of the graph). We are interested in the impact of the constraints on this bound. That is, given a graph G, we are interested in the largest k for which the coordination with constraints and preferences task in G is solvable for every preference multiset M of size at most k. We focus on asynchronous systems in which an arbitrarily large number of processes are subject to crash failures. Each process has its own private registers, and the processes communicate via read/write accesses to a shared memory.

1.2 Our results

We first focus on the problem for the n-node path P_n because it enables to prove a lower bound on the size of Hamiltonian graphs for which the coordination with constraints and preferences task is solvable. Interestingly, this lower bound is almost twice as large as the $2k - 1$ bound without constraints resulting from renaming or musical chairs. Specifically, we establish the following:

Theorem 1. *Let k be a positive integer. The smallest integer n for which the coordination with constraints and preferences task in P_n is solvable for k processes satisfies $n = 4k - 3$. As a consequence, if the coordination with constraints and preferences task in an n-node Hamiltonian graph G is solvable for k processes then $n \geq 4k - 3$.*

The lower bound on n is based on a reduction to impossibility results for musical chairs, i.e., renaming with initial preferences. The upper bound on n

comes from a wait-free algorithm, inspired from an optimal adaptive renaming algorithm, whose main lines are: (1) fix a maximum independent set I in P_n, (2) index the vertices of I from 1 to $2k-1$, and (3) run an optimal (adaptive) renaming algorithm on these indexes.

From this preliminary result on P_n, one may think that solving the coordination with constraints and preferences task in a graph G boils down to classical renaming once a maximum independent set in G is fixed. We show that this is *not* the case. In fact, even for an instance as simple as the n-node ring C_n, the problem becomes highly non trivial.

Theorem 2. *Let k be a positive integer. The smallest n for which the coordination with constraints and preferences task in C_n is solvable for k processes satisfies $4k-3 \leq n \leq 4k-2$.*

The lower bound is a consequence of Theorem 1 since C_n is Hamiltonian. A quite intriguing fact is that the wait-free algorithm derived from an adaptation of an optimal algorithm for classical renaming run on a maximum independent set of C_n does not match the lower bound, and is off by an additive factor $+1$. In fact, we prove that the true answer is probably the lower bound $4k-3$, which is shown to be tight for $k = 2$ and 3 agents, using ad hoc algorithms that are radically different from renaming algorithm.

We believe that the difference of 1 between the lower and upper bounds for C_n is certainly not anecdotal, but is the witness of a profound phenomenon that is not yet understood, with potential impact on classical renaming and musical chairs. The main outcome of this paper is probably the observation that "static" algorithms, i.e., algorithms based on *fixed* precomputed positions in the graph of constraints, might be sub-optimal by allocating less resources than the optimal. Our optimal ad hoc algorithms for coordinating two or three processes in the ring are not static, and the set of allocated resources output by these algorithms can form *any* independent set. The design of optimal "dynamic" (i.e., non static) algorithms for solving the coordination with constraints and preferences task appears to be a challenge, even in the specific case of the cycle C_n.

The enormous difficulty for asynchronous crash-prone processes to coordinate under constraints and preferences, even in graphs with arbitrarily large independent sets, is also illustrated by the case of the complete bipartite graph $K_{x,y}$ with $n = x + y$ nodes. We show that, although $K_{x,y}$ has very large independent sets (of size at least $\min\{x, y\}$), processes cannot coordinate *at all* in this graph.

Theorem 3. *Let x, y be positive integers. Coordination with constraints and preferences in the complete bipartite graph $K_{x,y}$ is unsolvable for more than one process.*

Finally, on the positive side, given any graph G, we can design an static algorithm ALG solving the coordination with constraints and preferences task in G. ALG is based on the novel notion of *k-admissible* independent sets, which may have its interest on its own: given $G = (V, E)$, an independent set I of G is k-admissible if for every $W \subseteq V$ of size at most $k-1$, we have $|I \setminus N[W]| \geq |I \cap W| + 1$

where $N[W]$ denotes the set of nodes at distance at most 1 from a node in W. We prove that among static algorithms, ALG is optimal, which completely closes the problem for static algorithms.

Theorem 4. *Let G be a graph, and k be a positive integer. Let I be a k-admissible independent set in G. Then,* ALG *instantiated with I solves the coordination with constraints and preferences task in G with k processes. Moreover, if G has no $(k+1)$-admissible independent set, then no static algorithms can solve the coordination with constraints and preferences task in G with more than k processes.*

1.3 Related Work

Since its introduction, the renaming problem has been extensively studied (see for example [5,10,19]). It was initially introduced as a *non-adaptive* problem in which processes just need to pick distinct output names in the space $[1, \ldots, M]$, where M is on function only on the total number of processes that might participate [4]. Several algorithms were proposed (e.g. [4,9,14]), and, as far as we know, all those initial algorithms are adaptive. Then the adaptive version of renaming was coined. The study of lower bounds for renaming have inspired new developments in topology techniques (see [17] for a detailed description). As explained above, the variant of renaming that is closest to this paper is the adaptive renaming version. This is the version we use to solve the coordination with constraints and preferences task on a graph with no edges.

Musical chairs [2,3] is a coordination problem on a stable graph in which each process starts with a initial vertex (chair) and processes are required to decide distinct vertices (chairs). The problem is studied in a model where the only communication between processes is an indication when two processes propose the same vertex. It has been shown that k processes can solve the problem only if the stable graph has at least $2k-1$ vertices. It has been also shown that musical chairs and adaptive renaming are equivalent problems.

Interestingly, the coordination with constraints and preferences is also related to *mobile computing* [12], where mobile entities (modeling physical robots, software agents, insects, etc.) must cooperate in order to solve tasks such as rendezvous, gathering, exploration, patrolling, etc. In particular, in the asynchronous look-compute-move model of mobile computation, the "look" operation is very similar to a "snapshot" operation in shared memory, and the "move" operation is very similar to the "write" operation. The major differences between the wait-free model of distributed computation and the look-compute-move model of mobile computation are (1) the presence of failures in the former, and (2) the fact that agents are moving in an anonymous graph in the latter.

2 Model and examples

2.1 Computational model

We consider the standard asynchronous wait-free read/write model with k processes, $p_1, \ldots, p_k$ [5,19]. Processes are asynchronous, communicate by writing and reading from a reliable shared memory, and any set of processes may crash. We assume, without loss of generality, that processes can read the whole shared memory in a single atomic snapshot [1].

Problems in the wait-free model are usually defined as *tasks*, where processes get *input values*, and decide after a bounded number of operations on *output values*, such that the decided values represent a valid configuration associated to the initial values of the execution for this task.

Without loss of generality, we can assume that algorithms solving tasks are in *normal form*, that is, are of the form of a loop consisting of (1) writing to the shared memory the local state of the process, (2) taking a snapshot, and (3) performing some local computation. The loop is executed until the process returns an output (i.e., decides).

2.2 Coordination with Constraints and Preferences (CCP)

The task *coordination with constraints and preferences* (or CCP for short) is instantiated by a fixed n-node graph $G = (V, E)$. The graph is modeling the *constraints*. It is supposed to be simple, i.e., without loops and multiple edges, but does not need to be connected. Each process p_i gets as input one vertex $u \in V$, called its initial private *preference*, and must eventually decide on a vertex $v \in V$. It is required that the decided vertices form an *independent set* of G, that is, no two processes decide the same vertex, and no two decided vertices belong to the same edge. It is also required that if the initial preferences form an independent set, then each process must decide its initial preference (enforcing the fact that processes cannot discard their preferences).

We are interested in computing, for every n-node graph G, the largest k such that CCP in G is wait-free solvable for k processes. Note that an algorithm solving CCP in G is given the full description of G a priori. Hence, there are no issues such as, e.g., breaking symmetry between nodes of G, even if G is vertex-transitive. (In particular, the nodes of G might be given labels from 1 to n, a priori, in a specific order which may facilitate the task for the processes).

2.3 Examples and Basic Observations

CCP is trivially solvable for one process in every graph, by selecting its initial preference as output vertex. Also, CCP is trivially not solvable in G for k processes if k exceeds the size of a maximum independent set. In fact, CCP is not solvable in G for k processes if k exceeds the size of the smallest maximal independent set. Indeed, let $I = \{u_1, \ldots, u_\ell\}$ be a smallest maximal independent set in G, and assume that ℓ processes $p_1, \ldots, p_\ell$ are given preferences in I (u_i to p_i for

$i = 1, \ldots, \ell$). In a wait-free execution in which only those ℓ processes participate, they must decide I. If another process $p_{\ell+1}$ "wakes up" after the ℓ processes have decided, there is no more room for $p_{\ell+1}$ to acquire a vertex, because I is maximal. This holds even if there exists another independent set I' larger than I, since the first ℓ processes have already terminated.

The following result is a direct consequence of [3] as Musical Chairs is exactly our problem on the stable graph.

Proposition 1. Let k be a positive integer. The smallest integer n for which the coordination with constraints and preferences task in the n-node stable graph is solvable for k processes satisfies $n = 2k - 1$.

Also, we have the following observation.

Proposition 2. Let $G = (V, E)$ be a graph, and $G' = (V, E')$ with $E' \subseteq E$ be a subgraph of G. If the coordination with constraints and preferences task is solvable for k processes in G, then it is solvable for k processes in G'.

As a consequence of the above two propositions, we get a general lower bound on the size of graphs in which the coordination with constraints and preferences task is solvable for k processes.

Corollary 1. *Let G be an n-node graph. If the coordination with constraints and preferences task in G is solvable for k processes then $2k - 1 \leq n$.*

3 The Case of Cyclic Constraints

Our first results concern simple non-trivial sets of constraints, namely the cases of the n-node paths and cycles, respectively denoted P_n and C_n. The case of the path is entirely solved by the following results, which establish Theorem 1:

Proposition 3. Let k be a positive integer. The smallest integer n for which the coordination with constraints and preferences task in the n-node path is solvable for k processes satisfies $n = 4k - 3$.

Proof. Let us assume, for the purpose of contradiction, that there is an algorithm $\mathcal{A}$ solving CCP in the n-node path for k processes with $n = 4k - 4$. Such a path has a maximum matching M of size $2k - 2$. $\mathcal{A}$ guarantees that, for every edge of M, at most one process acquires an extremity of that edge. $\mathcal{A}$ can be used as a subroutine to solve CCP on a stable graph of size $2k - 2$, as we show below.

We assume that the n-path is oriented from left to right and hence for each edge in M there is a *left* vertex and a *right* vertex (thus, in the path, not two left (right) vertices are adjacent). Also, each vertex v of the stable graph of size $2k-2$ is mapped to a unique edge $f(v)$ in M. To solve CCP on the stable graph, each process p_i with initial preference v, invokes $\mathcal{A}$ with the left vertex of $f(v)$ and decides $f^{-1}(e)$, where e is the edge in M containing the vertex that $\mathcal{A}$ outputs to p_i The resulting algorithm $\mathcal{A}$ solves CCP on the stable graph of size $2k - 2$

because if processes start with distinct vertices, then all of them invoke $\mathcal{A}$ with left vertices and hence, each process decide its initial preference. If processes start with conflicting initial preferences, then $\mathcal{A}$ outputs vertices that belong to distinct edges in M. This is a contradiction with Proposition 1 because M is of size $2k-2$.

We now describe an algorithm solving CCP in the n-node path for k processes with $n = 4k-3$. The algorithm is based on a maximum independent set I of size $2k-1$ in P_{4k-3}. That is, the nodes of P_{4k-3} are labeled off line as $(v_1, v_2, v_3, \ldots, v_{4k-3})$, and we define $I = \{v_1, v_3, v_5, \ldots, v_{4k-3}\}$. Essentially, the algorithm runs the textbook renaming algorithm of [5] on I, adapted to handle initial preferences. Indeed, selecting a node $w \notin I$ may block two positions in I (the two neighboring nodes of w). Nevertheless, there is still enough room to perform renaming, and hence to solve CCP. Indeed, let $N[w]$ be the closed neighborhood of w in P_n, i.e., the at most three nodes at distance at most 1 from w, and, for a set of nodes W, let $N[W] = \cup_{w \in W} N[w]$. In classical renaming, if W is the multiset of currently chosen names, there remain at least $2k - 1 - |W| \geq k$ available names to choose from. In the path, if W is the multiset of currently chosen nodes, there are only $|I \setminus N[W]|$ available nodes in I to choose from, and this number of available nodes can be less than k. However, the crucial observation is that $|I \setminus N[W]|| > |I \cap W|$ in the path P_{4k-3}, for any set of nodes W of size at most $k-1$. Hence, there are more free nodes in I than occupied nodes in I, and thus the idea is to perform the ranking of the renaming algorithm only on processes sitting on the occupied nodes of I. Since $|I \setminus N[W]|| > |I \cap W|$, this ranking is valid, that is, systematically provides a position in $I \setminus N[W]$. Termination follows from classical arguments by assuming, by way of contradiction, that some processes do not terminate, and then by considering the process p with lowest ID that does not terminate. Eventually, the rank r of p will remain forever the same, and no other processes that do not terminate will conflict with the rth node in the subset of nodes in I that are not conflicting with terminated processes. At this point, process p terminates. □

As a consequence of this result combined with Proposition 2, we get a general lower bound on the size of Hamiltonian graphs in which the coordination with constraints and preferences task is solvable for k processes. Interestingly, this bound is roughly twice as big as the bound for arbitrary graphs (cf. Corollary 1).

Corollary 2. *Let G be an n-node Hamiltonian graph. If the coordination with constraints and preferences task in G is solvable for k processes then $4k-3 \leq n$.*

The case of P_n has attracted our interest for it enables deriving bounds for Hamiltonian graphs. The case of the cycle C_n may seem to behave quite similarly as P_n. Surprisingly, this is not the case, as the wraparound constraint yields an interesting phenomenon, namely "static" solutions inspired from renaming algorithms such as the ones for the stable graph and the path are not anymore optimal in term of number of processes, and are off by an additive factor $+1$ from the optimal. More precisely, we show the following, which establishes Theorem 2:

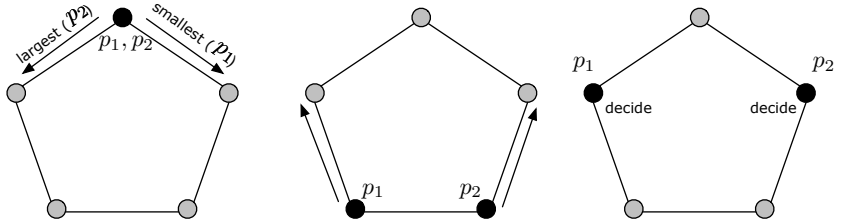

Fig. 1. *CCP algorithm for two processes in C_5: rules when two processes are executing*

Proposition 4. Let k be a positive integer. The smallest integer n for which the coordination with constraints and preferences task in C_n is solvable for k processes satisfies $4k - 3 \leq n \leq 4k - 2$.

Proof. The lower bound follows directly from Corollary 2. The upper bound is directly derived from the algorithm in the proof of Proposition 3 by fixing a maximum independent set of size $2k - 1$ in the cycle C_{4k-2}. The correctness of the algorithm follows from the same arguments as for the path P_{4k-3}. □

Interestingly, the lower bound $4k - 3$ is most probably the right answer, and not the upper bound $4k - 2$. At least, this is the case for small numbers of processes:

Proposition 5. The smallest integer n for which the coordination with constraints and preferences task in C_n is solvable for k processes satisfies $n = 4k - 3$ for $k = 2$ and $k = 3$.

Proof. The nodes of C_{4k-3} are sequentially labeled offline as $v_1, v_2, \ldots, v_{4k-3}$. This labeling induces a clockwise direction (increasing labels) and a counterclockwise direction.

The algorithm for two processes in C_5 is depicted on Figure 1, which represents the snapshot of a process, and the action to take (represented as arrows) based on this snapshot when 2 processes participate. There are three cases, depending on whether the two processes are currently occupying nodes at distance 0, 1, or 2. (Of course, if the snapshot reveals that the process is alone, then it decides the node that it currently occupies, i.e., its preferred node). If the snapshot reveals that the two processes occupy the same node, then the action depends on the ID: going clockwise for the process with smallest ID, and counterclockwise otherwise. If the snapshot reveals that the two processes occupy two neighboring nodes, then the action is: going away from the other node. Finally, if the snapshot reveals that the two processes occupy two nodes at distance 2, then the action is to decide the currently occupied node. One can check that this asynchronous algorithm terminates, and wait-free solves CCP.

A similar algorithm for three processes in C_9 can be derived. Due to lack of space, this algorithm is deferred to the full version of the paper. □

From these two cases, we conjecture that the smallest cycle C_n enabling to solve CCP is $n = 4k - 3$, for all $k \geq 2$. If this is correct, it means that optimality requires processes to coordinate in a more complex way than they do for renaming, in order to spread out optimally in the graph of constraints, and eventually occupy a large number of nodes. The independent set they will eventually agree on cannot be decided a priori, but the processes must agree *on line* in order to eventually decide an independent set that fits their initial preferences, the constraints and the uncertainty resulting for asynchrony and failures.

4 A Generic Algorithm

The algorithms inspired by the original algorithm of [5] used in the proofs of Theorems 1 and 2 to establish the upper bounds on the smallest integer n for which the coordination with constraints and preferences task in the n-node path and in the n-node cycle are *static* in the sense that they are aiming at deciding within a fixed independent set. More precisely:

Definition 1. Let G be a graph, and I be an independent set in G. An algorithm $\mathcal{A}$ solving the coordination with constraints and preferences task in G is *static with respect to* I if, for every execution of $\mathcal{A}$, and for every process p, if p does not decide its initial input, then it decides a vertex in I.

To establish Theorem 4, we present a *generic* static algorithm to solve CCP on every graph $G = (V, E)$, and prove that this algorithm is the best possible static algorithm in the sense that it maximizes the number k of processes for which CCP is solvable in G. The generic algorithm is instantiated with an a priori ordered independent set I. That is, $I = \{w_1, \ldots, w_{|I|}\}$, and this ordering of the nodes in I is know a priori to every process. The processes proceed in a sequence of (asynchronous) rounds. At each round, every process p_i proposes a current vertex denoted cur_i (the first proposal is the input u_i of processes p_i). Then, p_i checks whether there is a *conflict* with other proposals. In absence of conflict, p_i decides its current proposal cur_i. If there is a conflict, p_i computes a new proposal in I, and repeats. Hence, in particular, if a process sees no conflict in its initial proposal, then it stays there. Otherwise it will try a new proposal in I. The new proposal of p_i is computed within the "free space" that is defined as the maximal subset of I such that there is no conflict with other processes' proposals.

Algorithm 1 is the pseudocode of the generic algorithm. The algorithm uses a shared array *view*, accessed with write and snapshot operations, where each entry is initially $\perp$. For convenience, it is easier to consider the array *view* as a multiset of nodes. The local variable cur_i stores the current proposal of process p_i. For a set $W \subseteq V$, we denote by $N[W]$ the *closed neighborhood* of W, that is, for $w \in V$, $N[w] = \{w\} \cup \{v \in V : \{v, w\} \in E\}$, and $N[W] = \cup_{w \in W} N[w]$.

Note that if the initial preferences of participating processes are distinct and form an independent set, then Algorithm 1 guarantees that each process

Algorithm 1 $G = (V, E)$ is a graph, and I is an ordered independent set in G. Code for p_i.

function *IndependentSet*($u_i \in V$: initial preference of p_i)

```
1:  cur_i ← u_i
2:  loop
3:      write(cur_i)
4:      snapshot memory to get view = {cur_{j_1}, ..., cur_{j_r}}      ▷ multiset of r elements, for some r
5:      view' ← view \ {cur_i}                                          ▷ remove one occurence of cur_i from view
6:      if view' ∩ N[cur_i] = ∅ then                                    ▷ check for conflicts
7:          return cur_i                                                ▷ no conflict ⇒ decide cur_i
8:      else                                                            ▷ conflict detected ⇒ compute a new position
9:          free ← I \ N[view']                                         ▷ rule out conflicting vertices from I
10:         ℓ ← |{s : cur_{j_s} ∈ I and j_s < i}| + 1                   ▷ ranking on the currently occupied vertices of I
11:         cur_i ← ℓ^th element in free                                ▷ try the ℓ^th free node for the next round
12:     end if
13: end loop
```

decides on its initial preference. Note also that if two processes decide on vertices v_1, v_2, then $v_1 \neq v_2$ and $\{v_1, v_2\} \notin E$. However, for Algorithm 1 to function appropriately, we use it for a specific kind of independent sets, namely *admissible* independent sets, defined hereafter (see Figure 2 for an illustration):

Definition 2. Let k be a positive integer, and $G = (V, E)$ be a graph. An independent set I of G is *k-admissible* if for every $W \subseteq V$ of size at most $k - 1$, we have $|I \setminus N[W]| \geq |I \cap W| + 1$.

Notice that any k-admissible independent set I satisfies $|I| \geq 2k - 1$ (instantiate Definition 2 with $W \subseteq I$ of size $k - 1$). To establish Theorem 4, we first prove the following result.

Proposition 6. Let G be a graph, and k be a positive integer. Let I be a k-admissible independent set in G. Then, Algorithm 1 instantiated with I solves the coordination with constraints and preferences task in G with k processes.

Proof. We have seen that the safety conditions (i.e., respect of the preferences, and take decisions on an independent set) are satisfied. It just remains to show that the algorithm is valid (i.e., whenever a process detects a conflict on Line 6, it is able to compute a new consistent preference), and terminate. We first show validity.

Claim. For any process that is about to execute Line 11, $|\mathit{free}| \geq \ell$.

Consider a process p_i that is about to execute Line 11. Such a process p_i must have detected a conflict Line 6. Let *view* be the snapshot of p_i associated with this conflict, and $W = \mathit{view}'$. When p_i is about to execute Line 11, we have $\mathit{free} = I \setminus N[W]$. As there are at most k participating processes, it follows that $|W| < k$. Since I is k-admissible, we have $|I \setminus N[W]| \geq |W \cap I| + 1$, which implies $|\mathit{free}| \geq |W \cap I| + 1$. Moreover, p_i's ranking computed in line 10 is at most $|W \cap I| + 1$ because W does not contain p_i's preference, i.e., one ignores the nodes not in I when ranking. Hence $|\mathit{free}| \geq \ell$, and thus the algorithm is valid.

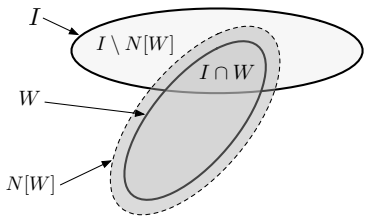

Fig. 2. *To be k-admissible, I must satisfy $|I \setminus N[W]| \geq |I \cap W| + 1$ for every W of size $\leq k - 1$.*

Claim. Algorithm 1 terminates.

Assume, for the sake of contradiction, that there is a run α in which some processes take an infinite number of steps without deciding a vertex. In this execution, let P be the set of all processes taking infinitely many steps, and let $p \in P$ be the process with minimum ID in P. Consider a suffix α' of α in which (1) all processes that do not run forever have stopped, and (2) all processes in P have already tried once to get a vertex in I, namely, they have executed Line 11 at least once. Note that such a suffix exists because every process that takes an infinite number of steps in α will see an infinite number of conflicts, and thus will eventually always execute Line 3 with its current vertex in I.

In α', the rank of p is fixed because, from there on, the set of processes that occupy vertices in I is fixed. Let r be the rank of p among the processes of α' with proposals in I, and let *good* be the set of vertices in I that do not conflict with preferences of stopped processes in α'. Eventually, there are no processes in P that are proposing one of the first $r - 1$ elements of *good*. Indeed, the rank of every process in P is at least r, and when a process $q \in P$ takes a snapshot, the set *free* that it computes satisfies *free* $\subseteq$ *good*. Thus, q proposes the x-th element in *free*, where $x \geq r$. Hence, this element cannot be one of the first $r - 1$ elements in *good*.

It follows that, in α', as soon as all running processes have seen each other, and have written at least twice, when these processes compute *free*, the first r elements are all elements of *good*, and only p will try to get the r-th element in *free*. When it does, it detects no conflict. Thus p can terminate the algorithm on Line 3, which yields to a contradiction. □

We now show the second part of Theorem 4, that is, Algorithm 1 is optimal on the number of processes, among static CCP algorithms. This is established thanks to the following result.

Proposition 7. Let G be a graph, k be a positive integer, and assume that G has no $(k + 1)$-admissible independent set. Then, no static algorithm can solve the coordination with constraints and preferences task in G with more than k processes.

The proof is based on the following claim, which is an interesting consequence of the Wait-free Computability Theorem of [18].

Claim. If there is a k-process CCP algorithm for G, then there exists a k-process CCP algorithm for G in which whenever a process sees only itself in its first snapshot, it immediately decides.

Proof (Sketch). The Asynchronous Computability Theorem ACT [18] states that a task is wait-free read/write solvable if and only if there is a chromatic subdivision of the input complex of the task (the simplicial complex represents all possible input configurations) with a coloring that agrees with the specification of the task.

Assume there is a k-process CCP algorithm for G, then let S be such a chromatic subdivision. Herlihy and Shavit proved that any chromatic subdivision can be approximated by an M-th standard chromatic subdivision [18], for some big enough M. This M-th standard chromatic subdivision is crucial because it directly implies a wait-free read/write protocol that solves the task that the coloring of S respects.

Now, the particular algorithms in the claim change the class of subdivisions we are dealing with. These subdivisions are very similar to the M-th standard chromatic subdivision with the difference that that after the 1-st standard chromatic subdivision, the corners of the subdivision are not changed, while the rest of the simplexes are subdivided in the same way. Let us call these subdivisions *M-solo chromatic subdivisions*. Considering this class of subdivisions, in which the diameter of these subdivisions shrinks as M grows, we can observe that, for a big enough M, say M_S, the M_S-solo chromatic subdivision approximates S. As explained above, this M_S-solo chromatic subdivision implies an algorithm that solves CCP for k processes for G in which whenever a process sees only itself in its first snapshot, it immediately decides. □

Proof (Proposition 7). For contradiction, assume there is a static CCP algorithm A on G for $k+1$ processes, with respect to a non $(k+1)$-admissible independent set I. By the claim above, we can assume that if a process sees only itself in its first snapshot, then it decides its initial value (recall that A can be assumed to be in normal form). Since I is not $(k+1)$-admissible, there exists a set $W \subseteq V$ with at most k vertices such that $|I \setminus N[W]| < |W \cap I| + 1$.

Claim. $W \cap I \neq \emptyset$.

To prove the claim, suppose by contradiction that $W \cap I$ is empty. It follows that $|W \cap I| + 1 = 1$, and thus $|I \setminus N[W]| = 0$, i.e., $I \setminus N[W]$ is empty as well. Thus, $I \cap N[W] = I$. Now, let us consider an execution α of A in which processes $p_1, \ldots, p_{|W|}$ start with preferences for distinct vertices in W, and just write their input in shared memory, i.e., they only perform one write operation in shared memory. Then, consider any extension β of α in which a process q starts with a preference to a vertex in I, runs alone (the p_i's from α do not take steps), and decides. Note that such a process exists since $|W| \leq k$ and A is for $k+1$ processes. Since A is static with respect to I, q decides a vertex $v \in I$. Let $v' \in W$ that is adjacent to v, whose existence is guaranteed by $I \cap N[W] = I$, and let q' be the process in α with input v'. Let α' be a reordering of α in which q' executes solo,

decides, then all other processes in α do their write operation. q' sees only itself in its first snapshot, and will thus decide its input. As q cannot distinguish α from α', β is also a valid extension of α'. In $\alpha'\beta$, process q decides v. This is a contradiction, since q' decides v' in this execution, which is in conflict with v. Thus, $W \cap I \neq \emptyset$, which completes the proof of the claim.

Let $\rho = |W \cap I|$. As proved above, $\rho > 0$. We have $|I \setminus N[W]| < |W \cap I| + 1 = \rho + 1$, and thus $|(I \setminus N[W]) \cup (W \cap I)| < 2\rho - 1$. Let us consider an execution α of A in which processes $p_1, \ldots, p_{|W \setminus I|}$ start with preferences for distinct vertices in $W \setminus I$, and they just write their input in the shared memory. Using the same indistinguishability argument as in the proof of the claim above, we can show that that there is no extension of α in which a process $q \notin \{p_1, \ldots, p_{|W \setminus I|}\}$ decides a vertex in $I \cap N[W]$. Thus, in every extension of α, such a process q decides its initial preference vertex, or a vertex in $(I \setminus N[W]) \cup (W \cap I)$. Let us then consider processes $q_1, \ldots, q_{\rho+1}$, all distinct from $p_1, \ldots, p_{|W \setminus I|}$ (note that these processes exist because $|W| \leq k$). In every extension of α in which $q_1, \ldots, q_{\rho+1}$ start with preferences in $(I \setminus N[W]) \cup (W \cap I)$, and these processes decide, they necessarily decide vertices in $(I \setminus N[W]) \cup (W \cap I)$. This implies that $|(I \setminus N[W]) \cup (W \cap I)| \geq \rho + 1$.

Now, from A and execution α, we construct a p-process algorithm A' that solves CCP in the stable graph with too few vertices as follows. In A', the shared memory has the state in α, and each process q_i, $1 \leq i \leq \rho + 1$, follows the same code as in A. To see that A' is correct, note that (1) if processes start with distinct vertices of $(I \setminus N[W]) \cup (W \cap I)$, then each process decides its input, since A is correct, and (2) if processes start with inputs in conflict, then they decide distinct vertices in $(I \setminus N[W]) \cup (W \cap I)$, as shown before. Now, A' solves CCP for $\rho + 1$ processes over the stable graph with vertex set $(I \setminus N[W]) \cup (W \cap I)$ (and no edges), which is impossible since this set has at most $2\rho - 2$ vertices, and, by Proposition 1, CCP is unsolvable for so few vertices. Therefore, assuming the existence of A yields a contradiction, which completes the proof. □

The following is an interesting consequence of Proposition 7 to the case of CCP in cycles.

Corollary 3. *If $\mathcal{A}$ is a wait-free algorithm solving the coordination with constraints and preferences task in C_{4k-3} for k processes then $\mathcal{A}$ cannot be static.*

A natural guess is that CCP can be solved in G for a number of processes that grows with the size of the smallest maximal independent set in G. Having this in mind, Theorem 4 may appear to be rather weak. Indeed, for instance, in complete bipartite graphs $K_{x,y} = (X \cup Y, X \times Y)$, with $x = |X|$, and $y = |Y|$, there are no 2-admissible independent sets. Thus our static algorithm, although optimal among static algorithms, cannot do better than solving CCP in $K_{x,y}$ for just one process! The truth is that our algorithm is not to be blamed. Indeed, the intuition that the number of processes that can be accommodated should grow with the size of the smallest maximal independent set is wrong, and *any* algorithm, not just static ones, cannot do better that solving CCP in $K_{x,y}$ for a single process only, as shown below (which establishes Theorem 3).

Proposition 8. Let x, y be positive integers. Coordination with constraints and preferences in $K_{x,y}$ is unsolvable for more than one process.

Proof. Recall that, in the *s-set agreement* task, each process proposes a value, and each correct process decides a proposed value so that the number of distinct decisions is at most s. We show that if algorithm $\mathcal{A}$ solves CCP in $K_{x,y}$ for k processes (hence either x or y is at least k), then there is an algorithm $\mathcal{B}$ that solves $\lceil \frac{k}{2} \rceil$-set agreement for k processes – yet, it is known that s-set agreement on k processes is unsolvable for any $1 \leq s \leq k-1$ and $k \geq 2$ (see [8,18,20]), a contradiction. In $K_{x,y} = (X \cup Y, X \times Y)$, X and Y are the unique two maximal independent sets. We say that processes $p_1, \ldots, p_{\lceil \frac{k}{2} \rceil}$ *belong* to X, and the remaining processes, $p_{\lceil \frac{k}{2} \rceil + 1}, \ldots, p_k$, *belong* to Y, in the following sense: to the processes $p_1, \ldots, p_{\lceil \frac{k}{2} \rceil}$, we assign pairwise distinct vertices $v_i \in X$ as preferences, $i = 1, \ldots, \lceil \frac{k}{2} \rceil$, and to the processes $p_{\lceil \frac{k}{2} \rceil + 1}, \ldots, p_k$, we assign pairwise distinct vertices $v_i \in Y$ as preferences, $i = \lceil \frac{k}{2} \rceil + 1, \ldots, k$. We use algorithm $\mathcal{A}$ solving CCP on this instance for construction an algorithm $\mathcal{B}$ solving $\lceil \frac{k}{2} \rceil$-set agreement.

A process p_i that belongs to X first announces its preference (considered as its proposal), and then invokes $\mathcal{A}$ using $v_i \in X$ as its input. If p_i gets assigned to a vertex of X in $\mathcal{A}$, then it decides v_i in $\mathcal{B}$, otherwise it decides any vertex $v_j \in Y$ with $j \in \{\lceil \frac{k}{2} \rceil + 1, \ldots, k\}$. A process p_i that belongs to Y proceeds similarly. That is, if p_i gets assigned to a vertex of Y in $\mathcal{A}$, then it decides its preference v_i in $\mathcal{B}$, otherwise it decides any vertex $v_j \in X$ with $j \in \{1, \ldots, \lceil \frac{k}{2} \rceil\}$. Since $\mathcal{A}$ is correct, and since there are no independent sets in $K_{x,y}$ that include vertices of both X and Y simultaneously, it follows that if a process gets assigned to a vertex of X (resp., Y) in $\mathcal{A}$, then every other process gets assigned to a vertex of X (resp., Y) as well. Therefore, in every execution of $\mathcal{B}$, processes either decide the preferences of the processes that belongs to X, or the preferences of the processes that belongs to Y, hence at most $\lceil \frac{k}{2} \rceil$ proposals are decided in $\mathcal{B}$. □

5 Conclusion

We have considered a generalization of renaming in graphs, in which deciding a node forbids others to use neighboring nodes. We proved a lower bound for Hamiltonian graphs, and provided optimal algorithms for 2 processes on a pentagon, and 3 processes on a nonagon. For the case where processes agree beforehand on a given maximal independent set, we designed optimal *static* algorithms for solving this problem. Static algorithms are however sub-optimal, as illustrated in the case of the rings. The design of optimal *dynamic* algorithms for solving the coordination with preferences and constraints tasks in graphs remains an open problem, even for rings.

References

1. Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, September 1993.

2. Yehuda Afek, Yakov Babichenko, Uriel Feige, Eli Gafni, Nati Linial, and Benny Sudakov. Oblivious collaboration. In David Peleg, editor, *Distributed Computing*, volume 6950 of *Lecture Notes in Computer Science*, pages 489–504. Springer Berlin Heidelberg, 2011.
3. Yehuda Afek, Yakov Babichenko, Uriel Feige, Eli Gafni, Nati Linial, and Benny Sudakov. Musical chairs. *SIAM Journal on Discrete Mathematics*, 28(3):1578–1600, 2014.
4. H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, 1990.
5. Hagit Attiya and Jennifer Welch. *Distributed Computing Fundamentals, Simulations, and Advanced Topics, Second Edition.* John Wiley and Sons, Inc., 2004.
6. Brenda S. Baker and Edward G. Coffman Jr. Mutual exclusion scheduling. *Theor. Comput. Sci.*, 162(2):225–243, 1996.
7. Hans L. Bodlaender and Klaus Jansen. Restrictions of graph partition problems. part I. *Theor. Comput. Sci.*, 148(1):93–109, 1995.
8. Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, New York, NY, USA, 1993. ACM.
9. Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC '93, pages 41–51, New York, NY, USA, 1993. ACM.
10. Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Comput. Sci. Rev.*, 5(3):229–251, August 2011.
11. Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. Scheduling*, 12(2):199–224, 2009.
12. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
13. Eli Gafni, Achour Mostéfaoui, Michel Raynal, and Corentin Travers. From adaptive renaming to set agreement. *Theoretical Computer Science*, 410(14):1328 – 1335, 2009. Structural Information and Communication Complexity (SIROCCO 2007).
14. Eli Gafni and Sergio Rajsbaum. Recursion in distributed computing. In Shlomi Dolev, Jorge Cobb, Michael Fischer, and Moti Yung, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 6366 of *Lecture Notes in Computer Science*, pages 362–376. Springer Berlin Heidelberg, 2010.
15. M. R. Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.*, 4(2):187–200, 1975.
16. Magnús M. Halldórsson, Guy Kortsarz, Andrzej Proskurowski, Ravit Salman, Hadas Shachnai, and Jan Arne Telle. Multicoloring trees. *Inf. Comput.*, 180(2):113–129, 2003.
17. Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology.* Morgan Kaufmann, 2013.
18. Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.
19. Michel Raynal. *Concurrent Programming: Algorithms, Principles, and Foundations.* Springer, 2013.
20. Michael Saks and Fotios Zaharoglou. Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.