The Offline Carpool Problem Revisited

Saad Mneimneh(✉) and Saman Farhat

Hunter College and The Graduate Center,
City University of New York (CUNY), New York, USA
saad@hunter.cuny.edu, sfarhat@gc.cuny.edu

Abstract. The carpool problem is to schedule for every time $t \in \mathbb{N}$ l tasks taken from the set $[n]$ $(n \geq 2)$. Each task i has a weight $w_i(t) \geq 0$, where $\sum_{i=1}^{n} w_i(t) = l$. We let $c_i(t) \in \{0, 1\}$ be 1 iff task i is scheduled at time t, where (carpool condition) $w_i(t) = 0 \Rightarrow c_i(t) = 0$.

The carpool problem exists in the literature for $l = 1$, with a goal to make the schedule fair, by bounding the absolute value of $E_i(t) = \sum_{s=1}^{t}[w_i(s) - c_i(s)]$. In the typical online setting, $w_i(t)$ is unknown prior to time t; therefore, the only sensible approach is to bound $|E_i(t)|$ at all times. The optimal online algorithm for $l = 1$ can guarantee $|E_i(t)| = O(n)$. We show that the same guarantee can be maintained for a general l. However, it remains far from an ideal $|E_i(T)| < 1$ when all tasks have reached completion at some future time $t = T$.

The main contribution of this paper is the offline version of the carpool problem, where $w_i(t)$ is known in advance for all times $t \leq T$, and the fairness requirement is strengthened to the ideal $|E_i(T)| < 1$ while keeping $E_i(t)$ bounded at all intermediate times $t < T$. This problem has been mistakenly considered solved for $l = 1$ using Tijdeman's algorithm, so it remains open for $l \geq 1$. We show that achieving the ideal fairness with an intermediate $O(n^2)$ bound is possible for a general l.

Keywords: Carpool problem · Fair scheduling · Graphs · Flows · Online and offline algorithms

1 Introduction

The carpool problem was first systematically studied by Fagin and Williams [1] to resolve issues of fairness related to the following scenario: There are n people. Each day a subset of these people will participate in a carpool and only one of them must be designated to drive. Fairness dictates that each person should drive a number of times (approximately) equal to the sum of the inverses of the number of people who showed up on the days that person participated in the carpool.

The problem has been generalized (see for instance [2,3]) by introducing weights: We schedule for every time $t \in \mathbb{N}$ one task taken from the set $[n]$

S. Mneimneh—Partially supported by the CoSSMO institute at CUNY.

S. Farhat—Supported by a CUNY Graduate Center Fellowship.

($n \geq 2$). Each task i has a weight $w_i(t) \geq 0$, where $\sum_i w_i(t) = 1$. We say that $c_i(t) \in \{0, 1\}$ is 1 iff i is the scheduled task at time t. Since exactly one task is scheduled at any time, $\sum_i w_i(t) = \sum_i c_i(t) = 1$, and $\sum_i E_i(t) = 0$. In addition, the following **carpool condition** is enforced:

Definition 1 (Carpool condition). $w_i(t) = 0 \Rightarrow c_i(t) = 0$ *for every* $i \in [n]$.

In effect, $w_i(t) > 0$ indicates the presence of the task at time t. Translating back to the original scenario, this is just to say a person must show up on the day he is the designated driver. When $w_i(t)$ is the same for every $i \in \{j | w_j(t) > 0\}$, we retrieve the special case introduced in [1].

We generalize the carpool problem further. Assume $l \in \mathbb{N}$ ($l \leq n$).

The Carpool Problem: Our version of the carpool problem is to schedule l tasks for every time $t \in \mathbb{N}$, where $\sum_{i=1}^{n} w_i(t) = \sum_{i=1}^{n} c_i(t) = l$ and $w_i(t) \leq 1$, and $c_i(t) \in \{0, 1\}$ is 1 iff task i is scheduled at time t, subject to the carpool condition.

The typical setting is online, i.e. $w_i(t)$ is unknown prior to time t, and one has to make immediate choices to construct a fair schedule by bounding the absolute value of $E_i(t) = \sum_{s=1}^{t} [w_i(s) - c_i(s)]$ at all times. It was shown in [3] that the online *greedy* algorithm that schedules at time t one (so $l = 1$) task $i \in \{j \in [n] | w_j(t) > 0\}$ such that $\theta_i(t) = \sum_{s=1}^{t-1} [w_i(s) - c_i(s)] + w_i(t)$ is maximized will achieve $|E_i(t)| = O(n)$ at all times. In the online setting, this (deterministic) bound is asymptotically optimal.

We consider the offline version of the carpool problem, where $w_i(t)$ is known in advance for all times $t < T$ for some fixed T. In this setting, the notion of fairness is strengthened to $|E_i(T)| < 1$. In other words, if T is the completion time of the schedule, each task i will have been served as closely as possible to its share $\sum_{s=1}^{T} w_i(s)$.

Definition 2 (Fair schedule). *A schedule is fair iff* $|E_i(T)| < 1$ *for every* $i \in [n]$*, where* $E_i(t) = \sum_{s=1}^{t} [w_i(s) - c_i(s)]$ *and*

$$c_i(t) = \begin{cases} 1 \; i \, \textit{is a scheduled task at time } t \\ 0 \; \textit{otherwise} \end{cases}$$

The above fairness property of carpool makes it attractive to many load balancing problems; however, in many applications, the correctness/guarantee of the scheduling algorithm will also require $|E_i(t)|$ to be bounded for all $t < T$ (see for instance [2]). We refer to this property in the offline context as *non-bursty*. While an online algorithm is naturally non-bursty, fairness as in Definition 2 alone may not be enough when moving to the offline version. Therefore, in addition to fairness, we will require the following:

Definition 3 (Non-bursty schedule). *A schedule is non-bursty iff* $|E_i(t)|$ *is bounded (independent of* t*) at all times* $t < T$ *for every* $i \in [n]$.

The problem of constructing an offline schedule that is fair and non-bursty was thought to be solved for $l = 1$. Section 2 covers the detail behind that misconception. For now, to further motivate our work, observe that the offline setting is important not only when the instance is known in advance, but also for recovery. For example, consider a scenario in which a backlog of T time steps has been created as a result of an interruption in the scheduler at time t_0. The scheduler looses all information and, therefore, if $|E_i(t_0)| < B$ for every $i \in [n]$, B will be added to whatever bound is achieved after recovery. When scheduling is resumed at time t, the scheduler works on the backlog starting with tasks presented at $t_0, t_0 + 1, \ldots$. It is then only logical to make use of the information in the backlog. As such, an offline algorithm offers a chance to maintain the same guarantee prior to the failure: $|E_i(t + T - 1)| < B + 1$, as opposed to the $B + O(n)$ bound of an online algorithm, which adds another $O(n)$.

2 Related Work

The carpool problem exists in the literature only for $l = 1$. The work in [1,3,4,6,9] (and their references) provide extensive treatment of the online setting. But the offline version of the carpool problem has not been explicitly mentioned in the literature. An exception lies in a few instances, e.g. [5,7], that model the offline carpool problem for $l = 1$ as a flow problem, as in Fig. 1, where $m_i = \lceil \sum_{t=1}^{T} w_i(t) \rceil$, and an edge from vertex i to vertex t exists iff $w_i(t) > 0$.

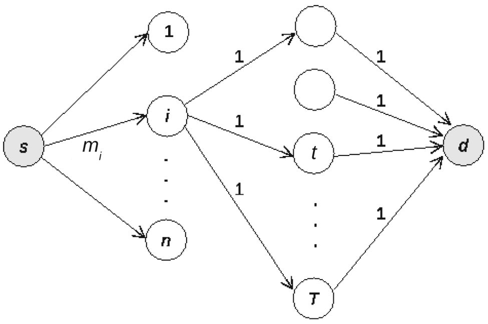

Fig. 1. The carpool problem for $l = 1$ as a flow.

The theory of flows, integer programs, and their linear program relaxations suggest that an integer maximum flow of value T exists and corresponds to a valid schedule in which $|E_i(T)| < 1$ for every $i \in [n]$, see for instance [5,8]. In addition, it is not hard to generalize the flow of Fig. 1 to handle any l. But this does not immediately yield an obvious bound on $|E_i(t)|$ for every $t < T$.

A short note in [4] (in the context of a problem called vector rounding) mistakenly claims that Tijdeman's algorithm [10] solves the offline carpool problem (for $l = 1$). Indeed, the carpool problem is closely related to the chairman assignment problem popularized by Tijdeman in [10], where tasks are persistent (a task can be scheduled at any time) so the carpool condition $w_i(t) = 0 \Rightarrow c_i(t) = 0$ is **dropped**.

For the chairman assignment problem, Tijdeman's algorithm guarantees $|E_i(t)| < 1$ at all times. To schedule a task at time t, the algorithm considers the set of tasks that satisfy $\theta_i(t) = \sum_{s=1}^{t-1}[w_i(s) - c_i(s)] + w_i(t) \geq 1/(2n-2)$ (this set must be non-empty), finds the smallest $t_0 \leq T$ such that $\theta_i(t) + \sum_{s=t+1}^{t_0} w_i(s) \geq 1 - 1/(2n-2)$ for some i in the set, and makes $c_i(t) = 1$ (if no such t_0 exists, the algorithm schedules any task i). There is no guarantee that $w_i(t) > 0$ for the scheduled task i. Figure 2 shows a counterexample with $n = 4$ and $T = 5$.

task	1	2	3	4	5
1	0	1/12	1/6	**0**	4/5
2	1/12	0	1/6	1/100	0
3	1/2	**1/2**	1/4	1/100	0
4	**5/12**	5/12	**5/12**	98/100	1/5

Fig. 2. Tijdeman's algorithm fails for carpool: weights in bold represent the schedule of tasks over time. Task 1 is scheduled at time $t = 4$ so $c_1(4) = 1$, but $w_1(4) = 0$.

Following [4], we see no subsequent attempt in the literature to construct an offline schedule that is both fair and non-bursty. The generalization to l tasks is absent as well.

3 Our Contribution

We believe this is the first explicit treatment of the offline carpool problem (also generally in the context of scheduling multiple tasks, i.e. $l > 1$). We show that achieving fairness ($|E_i(T)| < 1$ according to Definition 2) with an intermediate $O(n^2)$ bound (Definition 3) is possible. Our offline scheduling algorithm has a running time with a linear dependence on T. In order to achieve the $O(n^2)$ intermediate bound, we rely on a generalization of the online *greedy* algorithm described in [3] to handle multiple tasks. The schedule obtained by this algorithm is then used to transform a fair schedule to a fair and non-bursty one.

4 Generalizing the Online Algorithm

The online *greedy* algorithm described in [3] for $l = 1$ schedules at time t one task $i \in \{j \in [n] | w_j(t) > 0\}$ such that $\theta_i(t) = \sum_{s=1}^{t-1}[w_i(s) - c_i(s)] + w_i(t)$ is maximized, and achieves $|E_i(t)| < (n-1)/2$ at all times. We will describe an online algorithm with the same guarantee for a general l, called *l-greedy*.

The *l-greedy* algorithm schedules at time t the l tasks corresponding to the l largest elements in $\{\theta_i(t) | w_i(t) > 0\}$. This online algorithm can run offline in $O(lnT)$ time. We will show below that this algorithm has the same guarantee as *greedy*.

Theorem 1. *The l-greedy algorithm has at worst the same guarantee as greedy.*

Proof. We construct an imaginary instance of *greedy* that schedules the same tasks. Without loss of generality, assume that $X = \{1, 2, \ldots, l\}$ is the set of tasks scheduled at time t by the *l-greedy* algorithm. Starting from $t = 1$, divide each t into l times $t_1, \ldots, t_l$, and define $(1 \leq i, j \leq l)$

$$w_i^{'}(t_k) = \begin{cases} w_i(t) & i = k \\ 0 & \text{otherwise} \end{cases}$$

For all tasks $i \notin X$, define $w_i^{'}(t_k)$ such that $\sum_{k=1}^{l} w_i^{'}(t_k) = w_i(t)$ by dividing $\sum_{i \notin X} w_i(t)$ arbitrarily among $t_1, \ldots, t_l$ to make the sum of the weights for each time t_k equal to 1. Figure 3 illustrates this construction.

t	t_1	t_2	$\ldots$	t_l
$w_1(t)$	$w_1(t)$	0		0
$w_2(t)$	0	$w_2(t)$		0
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$w_l(t)$	0	0		$w_l(t)$
$w_{l+1}(t)$	$w_{l+1}^{'}(t_1)$	$w_{l+1}^{'}(t_2)$		$w_{l+1}^{'}(t_l)$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$w_n(t)$	$w_n^{'}(t_1)$	$w_n^{'}(t_2)$		$w_n^{'}(t_l)$

Fig. 3. Constructing the *greedy* instance. The weights as seen by the *l-greedy* algorithm are shown on the left, where $X = \{1, 2, \ldots, l\}$ are the scheduled tasks at time t. The weights as seen by *greedy* at times $t_1, \ldots, t_l$ are shown on the right, $w_i^{'}(t_i) = w_i(t)$ for $i \in X$, $\sum_{k=1}^{l} w_i^{'}(t_k) = w_i(t)$, and $\sum_{i=1}^{n} w_i^{'}(t_k) = 1$.

Consider $\theta_i^{'}(t_k)$ as seen by a *greedy* algorithm now acting on $t_1, \ldots, t_l$. Suppose that *greedy* has been scheduling the same tasks as *l-greedy*. If $i \in X$, then $\theta_i^{'}(t_i) = \theta_i(t)$; if $i \notin X$, then $\theta_i^{'}(t_k) \leq \theta_i(t)$. Therefore, $\theta_i^{'}(t_i)$ is the largest for t_i among all tasks j with $w_j^{'}(t_i) > 0$. So *greedy* will schedule for $t_1, \ldots, t_l$ exactly the tasks in X. □

We conclude that the *l-greedy* algorithm achieves the same guarantee as *greedy*, namely $|E_i(t)| = O(n)$ for every $i \in [n]$ and $\sum_{i=1}^{n} |E_i(t)| = O(n^2)$.

5 A Fair Schedule

As we mentioned in Sect. 2, it is possible to use flows to prove the existence of (and obtain) a fair schedule. But for completeness, we describe in this section a canonical algorithm to transform any arbitrary schedule, given by $c_i(t)$ for $i \in [n]$ and $t \leq T$, to a fair schedule (which will also serve as a proof for the existence of such a schedule). We construct a directed multigraph $G = (\mathbb{V} = [n], \mathbb{E})$ such that $(i, j)_t \in \mathbb{E}$ is an edge from i to j iff $c_i(t) = 1$ and $c_j(t) = 0$ and $w_j(t) > 0$. Such an edge means that task i is scheduled at time t but task j could be scheduled instead.

A path $\{(i_1, i_2)_{t_1}, (i_2, i_3)_{t_2}, \dots, (i_r, i_{r+1})_{t_r}\}$ with $E_{i_1}(T) < 0$ and $E_{i_r}(T) > 0$ represents a way to modify the schedule: make $c_{i_1}(t_1) = 0$ and $c_{i_2}(t_1) = 1$, $c_{i_2}(t_2) = 0$ and $c_{i_3}(t_2) = 1$, ... , $c_{i_r}(t_r) = 0$ and $c_{i_{r+1}}(t_r) = 1$. After making the changes, update the edges of the multigraph accordingly to reflect the new schedule. This will increase $E_{i_1}(T)$ by 1 and decrease $E_{i_{r+1}}(T)$ by 1. Therefore, if $E_{i_1}(T) \leq -1$ and/or $E_{i_{r+1}}(T) \geq 1$, the schedule is improved (by decreasing $|E_{i_1}(T)|$ and/or $|E_{i_{r+1}}(T)|$). Below we show that we can always make such improvements until the schedule becomes fair.

Lemma 1. *If $E_i(T) < 0$ ($E_i(T) > 0$), there is a path in G from i to some j (from some j to i) with $E_j(T) > 0$ ($E_j(T) < 0$). This path may be used to improve the schedule as stated above.*

Proof. We prove the case when $E_i(T) < 0$, the second case is symmetric. So assume $E_i(T) < 0$ and let the set A consist of all vertices j reachable from i such that $E_j(T) \leq 0$. The proof is by contradiction, so we can assume that there are no outgoing edges from A to the rest of the multigraph. Since $i \in A$, we know that $E = \sum_{j\in A} E_j(T) = E_i(T) + \sum_{j\in A, j\neq i} E_j(T) < 0$. Now consider the set $B = \{t | c_j(t) = 1 \text{ for some } j \in A\}$. Since A has no outgoing edges, if $j \notin A$ and $t \in B$, then $c_j(t) = 0 \Rightarrow w_j(t) = 0$ (so $w_j(t) - c_j(t) \leq 0$). Therefore, it should be clear that $e = \sum_{j\in A}\sum_{t\in B}[w_j(t) - c_j(t)] \geq \sum_{j\in A}\sum_{t\in B}[w_j(t) - c_j(t)] + \sum_{j\notin A}\sum_{t\in B}[w_j(t) - c_j(t)] = \sum_j \sum_{t\in B}[w_j(t) - c_j(t)] = \sum_{t\in B}\sum_j [w_j(t) - c_j(t)] = 0$. But $E = e + \sum_{j\in A}\sum_{t\notin B}[w_j(t) - 0] \geq 0$, a contradiction. □

Theorem 2. *There exists a fair schedule for every instance of the offline carpool problem.*

Proof. If a schedule is not fair, then there exists a task i such that $E_i(T) \geq 1$ or $E_i(T) \leq -1$. Therefore, one could apply Lemma 1. When we consider the sum

$$S = \sum_{i\in[n]} \lfloor |E_i(T)| \rfloor$$

we observe that it will decrease by either 1 or 2 after each iteration of Lemma 1: $E_i(T) \geq 1$ decreases by 1 for some i, or $E_j(T) \leq -1$ increases by 1 for some j, or both. Therefore, the sum will eventually reach 0. When this happens, $|E_i(T)| < 1$ for every $i \in [n]$ and the schedule is fair. □

Since we can write the above sum in two parts

$$S = \sum_{i\in\{j|E_j(T)>0\}} \lfloor E_i(T) \rfloor - \sum_{i\in\{j|E_j(T)<0\}} \lceil E_i(T) \rceil$$

and the first part is at most $\sum_{i,t} w_i(t) = lT$ and the second part is at least $-\sum_{i,t} c_i(t) = -lT$, we conclude that $S = O(lT)$. This is an upper bound on the number of iterations needed to modify any arbitrary schedule. But given a specific initial schedule, one could refine this bound. For the schedule obtained by the *l-greedy* algorithm of Sect. 4, $S = O(n^2)$. Therefore, we have $O(n^2)$ iterations, each will process the multigraph starting from some vertex in $O(lnT)$ time (the size of the multigraph) leading to an $O(ln^3T)$ time algorithm.

6 A Non-bursty Schedule

We now describe a canonical algorithm to transform a fair schedule, given by $c_i(t)$ for $i \in [n]$ and $t \leq T$, to a non-bursty schedule without affecting its fairness. For this, we assume the existence of an auxiliary non-bursty schedule (but not necessarily fair). Let S_t^{orig} and S_t^{aux} be the set of tasks scheduled at time t by those schedules respectively. We construct a directed multigraph $G = (\mathbb{V} = [n], \mathbb{E})$ such that $(i,j)_t \in \mathbb{E}$ is an edge from i to j if $i \in S_t^{orig} - S_t^{aux}$ and $j \in S_t^{aux} - S_t^{orig}$. Such an edge represents a discrepancy between the two schedules at time t; the original schedules task i but the auxiliary schedules task j. We make all such edges $(i,j)_t$ for a given t form a maximal matching in $[n]$ (so there are exactly $|S_t^{orig} - S_t^{aux}| \leq l$ of them). Consequently, the out-degree of vertex i is the number of times task i is scheduled by the original schedule but not by the auxiliary and, similarly, the in-degree of vertex i is the number of times task i is scheduled by the auxiliary schedule but not by the original.

A cycle $\{(i_1,i_2)_{t_1}, (i_2,i_3)_{t_2}, \ldots, (i_r,i_1)_{t_r}\}$ represents a way to bring closer the two schedules by modifying the original schedule as follows: make $c_{i_1}(t_1) = 0$ and $c_{i_2}(t_1) = 1$, $c_{i_2}(t_2) = 0$ and $c_{i_3}(t_2) = 1$, ... , $c_{i_r}(t_r) = 0$ and $c_{i_1}(t_r) = 1$. After making these changes, eliminate the cycle from the multigraph. The time needed to eliminate all cycles is $O(n + lT)$ (the size of the multigraph), and that's when we obtain the modified schedule.

Given the modification to the original schedule as described above, we will use E^{mod} and E^{aux} to refer to these quantities in the modified and the auxiliary schedules, respectively. Observe that $E_i^{mod}(T) = E_i(T)$ is kept unchanged for every $i \in [n]$. Since the auxiliary schedule is non-bursty, one would expect that the original schedule will be transformed as such. Below we quantify this intuition.

Let $H_1 = (\mathbb{V} = [n], \mathbb{E}_1)$ be the subgraph of G obtained by the elimination of all the cycles in G. This multigraph can be converted into a simple weighted directed acyclic graph (DAG) $H_2 = (\mathbb{V} = [n], \mathbb{E}_2)$ such that $e = (i,j) \in \mathbb{E}_2$ is an edge from i to j iff $(i,j)_t \in \mathbb{E}_1$ for some t with weight $w(e) = |\{t | (i,j)_t \in \mathbb{E}_1\}|$. Define $w(i,j) = w(e)$ if $e = (i,j) \in \mathbb{E}_2$ and $w(i,j) = 0$ otherwise. Observe that $\sum_j w(i,j)$ is now the number of times task i is scheduled by the modified schedule but not by the auxiliary and, similarly, $\sum_j w(j,i)$ is the number of times task i is scheduled by the auxiliary schedule but not by the modified.

Lemma 2. *If the auxiliary schedule (non-bursty) guarantees an intermediate bound $|E_i^{aux}(t)| < f(n)$, then*

$$\left| \sum_{j \in [n]} w(i,j) - \sum_{j \in [n]} w(j,i) \right| = E_i^{mod}(T) - E_i^{aux}(T) < 1 + f(n)$$

for every $i \in [n]$.

Proof. Given the above interpretation of $\sum_j w(i,j)$ and $\sum_j w(j,i)$, observe that $\sum_j [w(i,j) - w(j,i)] = E_i^{mod}(T) - E_i^{aux}(T)$. Therefore, the result is immediate because the modified schedule is fair (so $|E_i^{mod}(T)| < 1$) and the auxiliary schedule satisfies $|E_i^{aux}(T)| < f(n)$. □

Lemma 3. *If the auxiliary schedule (non-bursty) guarantees an intermediate bound* $|E_i^{aux}(t)| < f(n)$, *then*

$$\sum_{j\in[n]} w(i,j) < \frac{n}{2}[1+f(n)]$$

$$\sum_{j\in[n]} w(j,i) < \frac{n}{2}[1+f(n)]$$

for every $i \in [n]$.

Proof. Given vertex i, divide the DAG into m sets of vertices $V_1, \dots, V_r, \dots, V_m$ such that $V_r = \{i\}$ and all edges are from V_{k-1} to V_k for $k = 1, \dots, m$ ($V_0 = \emptyset$), as shown in Fig. 4.

Let

$$I_k = \sum_{i\in V_{k-1}, j\in V_k} w(i,j)$$

and observe that

$$I_{k+1} - I_k < |V_k|[1+f(n)]$$

by Lemma 2. Therefore, by summing up these inequalities over $k = 1, \dots, r-1$, we get

$$I_r < (|V_1| + \ldots + |V_{r-1}|)[1+f(n)]$$

By a symmetric argument, we also have

$$I_{r+1} < (|V_{r+1}| + \ldots + |V_m|)[1+f(n)]$$

and invoking Lemma 2 on $I_r - I_{r+1}$ gives

$$I_r < (|V_{r+1}| + \ldots + |V_m| + 1)[1+f(n)]$$

Finally,

$$2I_r < (|V_1| + \ldots + |V_{r-1}| + |V_{r+1}| + \ldots + |V_m| + 1)[1+f(n)] = n[1+f(n)]$$

and the same is true for I_{r+1} by symmetry. This concludes the proof because

$$\sum_{j\in[n]} w(i,j) = I_{r+1} \text{ and } \sum_{j\in[n]} w(j,i) = I_r$$ □

Lemmas 2 and 3 can be trivially generalized by changing $1 + f(n)$ to $B + f(n)$ if the fair schedule is replaced by a weaker schedule that guarantees $|E_i(T)| < B$. Given Lemmas 2 and 3, we just proved the following theorem.

Theorem 3. *If the auxiliary schedule (non-bursty) guarantees an intermediate bound* $|E_i^{aux}(t)| < f(n) = \Omega(1)$, *then the modified schedule (fair) satisfies* $|E_i^{mod}(t)| = O(nf(n))$ *at all times for every* $i \in [n]$.

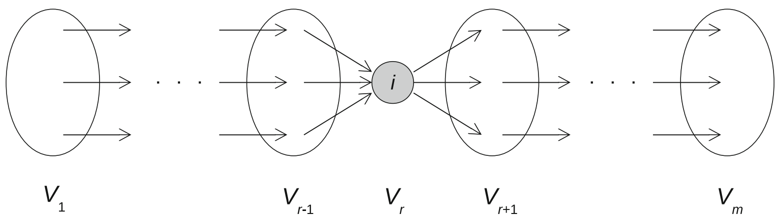

Fig. 4. Schematic illustration for the proof of Lemma 3.

7 The Final Algorithm

We now describe an offline algorithm for the carpool problem that is fair ($|E_i(T)| < 1$) and non-bursty with an $O(n^2)$ intermediate bound ($|E_i(t)| = O(n^2)$ for all $t < T$). The algorithm has a running time with a linear dependence of T, which is a nice feature.

1. Obtain an initial schedule using the *l-greedy* algorithm of Sect. 4.
2. Modify the schedule to be fair using the canonical algorithm of Sect. 5.
3. Modify the schedule to be non-bursty (and fair) using the canonical algorithm of Sect. 6, and the schedule of the *l-greedy* algorithm as the auxiliary schedule.

Since the *l-greedy* algorithm guarantees $|E_i(t)| = O(n)$ at all times, Theorem 3 implies that the above algorithm is non-bursty with an intermediate bound of $O(n^2)$.

8 Conclusion

The carpool problem is a scheduling problem where every task must receive its fair share, but may not be available at all times. We believe this is the first explicit treatment of the offline version of the carpool problem, which has been only studied in the online setting. In a typical offline setting, the goal would be to guarantee fairness by the schedule completion time T while avoiding an unbounded deviation from fairness at all times $t < T$. We achieved this goal by combining offline and online algorithms.

References

1. Fagin, R., Williams, J.H.: A fair carpool scheduling algorithm. IBM J. Res. Dev. **27**(2), 133–139 (1983)
2. Mneimneh, S.: Load balancing in a switch without buffers. In: IEEE Workshop on High Performance Switching and Routing, Poznan (2006)
3. Coppersmith, D., Nowicki, T., Paleologo, G., Tresser, C., Wu, C.W.: The optimality of the online greedy algorithm in carpool and chairman assignment problems. ACM Trans. Algorithms, **7**(3), Article 37, July 2011
4. Ajtai, M., Aspnes, J., Naor, M., Rabini, Y., Schulman, L.J., Waarts, O.: Fairness in Scheduling. J. Algorithms **29**(2), 306–357 (1988)

5. Naor, M.: How to Carpool Fairly. http://www.wisdom.weizmann.ac.il/naor/PAPERS/carpool_fair.pps
6. Naor, M.: On fairness in the carpool problem. J. Algorithms **55**(1), 93–98 (2005)
7. Williamson, D.: Lecture Notes on Network Flows, Chapter 3. http://people.orie.cornell.edu/dpw/techreports/cornell-flow.pdf
8. Havet, F.: Combinatorial Optimization, Chapter 11 on Fractional Relaxation. http://www-sop.inria.fr/members/Frederic.Havet/
9. Boavida, J.B., Kamat, V., Nakum, D., Nong, R., Wu, C.W., Zhang, X.: Algorithms for the Carpool Problem. http://www.ima.umn.edu/2005-2006/MM8.9-18.06/activities/Wu-Chai/team6_rep.pdf
10. Tijdeman, R.: The chairman assignment problem. Discrete Math. **32**, 323–330 (1980)