Determining Resource Needs of Autonomous Agents in Decoupled Plans

Jasper Oosterman a  Remco Ravenhorst a  Pim van Leeuwen b
Cees Witteveen a

a Delft University of Technology, Algorithmics group, Delft
b National Aerospace Laboratory NLR, Amsterdam

Abstract
At airports, the turnaround process of aircraft is crucial for a timely and efficient handling of air traffic. During this process, a number of services need to be provided during the time the aircraft is at the gate: e.g., de-boarding, cleaning, catering, fuelling, and boarding. These services are provided by different service providers (agents), who have to coordinate their activities in order to respect the turnaround timeslot, the required service times and existing regulations. These temporal dependencies between services can be planned, but if disruptions occur re-planning is complex and often impossible. For this reason, in previous research a methodology and prototype have been devised to simplify the planning process by decoupling the overall plan into local plans for each agent. This paper builds on this research and introduces an important extension to the existing methodology and prototype: a new algorithm is introduced to take into account the minimal number of resources each service provider will need to accomplish its task.

1 Introduction
The central problem we discuss in this paper can be stated as follows. Given a specification $C$ of temporal constraints for a set $A$ of activities of a group of autonomous agents, (i) how to obtain for each agent $i$ a set $C_i$ of local constraints for its activities $A_i \subseteq A$, such that agent $i$ can schedule its activities independently from the others, and (ii) how to provide each agent $i$ with an estimation of the number of resources it will need to carry out its set of activities. Essentially this problem can be viewed as a coordination problem since it requires the establishment of local temporal constraints for each agent in such a way that, while each agent can choose its own schedule and resource plan independently from the others, the feasibility of the overall solution is still ensured.

In previous research, a methodology and prototype have been devised to solve this coordination problem applied to airport planning problems [9]. In a nutshell, in this research airport activities and their constraints were represented as a Simple Temporal Network (STN) and the temporal decoupling algorithm introduced by Hunsberger [5] was applied in order to break it up into several local (i.e. agent specific) temporal specifications. To illustrate the relevance of this approach, the turnaround process of aircraft was chosen as an application domain. The current paper builds on this previous work, and re-uses the application domain chosen in [9]. Of course, the results of the approach described here can be applied more generally.

In this paper, two important extensions are introduced to the existing decoupling methodology and turnaround prototype developed in [9]. First, a new algorithm is introduced to take into account the capacity - in terms of resources - of each ground handling agent for accomplishing its task. Second, before determining the order of tasks and consequently the capacity, we encompass the travel time between gates for each agent. This makes the application much more realistic, since the actual travel times between gates strongly influence the required capacity.

The turnaround process at airports concerns all activities that have to take place after an aircraft has arrived at its gate (the on-block time) until it departs from the gate (the off-block time). During this time passengers have to de-board and board, the aircraft has to be fuelled, catered and cleaned, technical inspection and maintenance have to be performed, amongst other things. These activities have to be performed by several autonomous agents (service providers) and are highly dependent on one another. For example, fuelling is only allowed after all passengers have deboarded and before new passengers have boarded.
Currently, for every type of aircraft, a turnaround plan is devised specifying for each service (catering, cleaning, boarding, fuelling, etc.) the exact timeslot in which it should be performed. The collection of these plans constitutes the overall turnaround plan for an airport for a specific day. An example high-level entry of such a turnaround plan is:

\[(\text{KL1857, C06, 12:00, 13:28})\]

where the flight number, the gate, and the on- and off-block time of a specific aircraft, respectively, is specified. At a lower level, all services that need to be planned between the on- and off-block times are listed, taking the temporal dependencies and minimum and maximum duration of each service into account.

To represent such a turnaround plan, often Simple Temporal Networks (STNs) are used [3]. A STN is a compact representation of all the (temporal) relationships between activities that need to be planned. In the aforementioned work [9] [8], a high-level STN was constructed of the overall domain. In a second step, a special Temporal Decoupling technique [5] was used to split the high-level STN $S$ in a collection $S_{i=1}^{n}$ of sub STNs - one per service provider $i$. The outcome is that each of the STNs $S_i$ contains a specification of the activities and temporal constraints for each service provider $i$. This service provider then is able to schedule its own activities without being forced to take into account the decisions of other agents: the temporal decoupling ensures that every combination of individually constructed schedules can always be merged into a global conflict-free schedule for the original global STN.

Although this technique offers the significant advantage of autonomous scheduling capability to the agents, the individual STNs do not provide information about the number of resources a service provider requires for carrying out its temporal plan in the STN. For example, from the global STN, a local STN can be decoupled for the fuelling service provider. Such a plan contains a specification of all $n$ fuelling activities the service provider has to perform on a certain day. In general the number $m$ of fuelling cars the company has will be significantly less than $n$ and the fuelling company will be interested in a detailed specification of the minimum number of resources (e.g., vehicles) needed to perform all the activities. Moreover, this vehicle will need a specific amount of time to reach gate $y$ after having fuelled an aircraft at gate $x$. Therefore, for each pair of gates $x$ and $y$ the minimum travel times $t(x,y)$ needs to be specified, leading to an additional set of temporal constraints specified in the local STN of the fuelling company.

Our resource consumption problem then can be stated as follows:

*Given a STN containing the temporal constraints for $n$ activities taking place at $m$ different locations and requiring one type of resource together with a specification of the travel times between the resource application locations, how can one determine the minimum number of resources needed to perform all activities satisfying the constraints?*

To answer this question, in Section 2 the necessary background on STNs and the Temporal Decoupling method is discussed. In Section 3, the algorithm to solve the resource consumption problem is detailed. Finally, in Section 4 some possible extensions of this solution are discussed.

## 2 Background

For each aircraft serviced at an airport a turnaround plan exists that specifies which activities (boarding, fuelling, cleaning, etc.) need to be performed when during the time the aircraft is at the gate/stand. In this section we will use a Simple Temporal Network (STN) to model a collection of turnaround plans and discuss the properties of such a STN in more detail. Next, we discuss a decoupling method that enables us to split a given STN in independent sub-STNs such that independently obtained solutions to these sub-STNs can be simply merged to obtain a solution for the original STN. The decoupling method will be used to split the global STN representing the total collection of turnaround plans into sub-STNs for each of the agents, allowing them to construct their operational plans independently of each other without violating any of the original global constraints.

### 2.1 Simple Temporal Networks

To model the turnaround plans we use a Simple Temporal Network (STN) [3]:

**Definition 2.1** A Simple Temporal Network $S$ is a tuple $\langle X, C \rangle$ where $X$ is a collection $\{x_0, \ldots, x_n\}$ of time variables, and $C$ is a finite collection of constraints over these variables. Each constraint $c \in C$ is of the...
form $c_{ij} : x_j - x_i \leq b_{ij}$, for some $b_{ij} \in \mathbb{Z}$. The variable $x_0$ represents a special fixed time value, the temporal referential point, taking the value 0.

To use STNs to specify temporal constraints on activities, every activity $\alpha$ is represented by two time variables (events) $x_i$ and $x_{i+1}$ indicating respectively the starting and the finishing time of $\alpha$.

In a STN $S$, for every pair of variables $x_i, x_j \in X$ there is a constraint $c_{ij} : x_i - x_j \leq b_{ij}$ and a constraint $c_{ji} : x_j - x_i \leq b_{ji}$. If these constraints are combined we obtain the interval constraint $-b_{ji} \leq x_i - x_j \leq b_{ij}$, also written as $I_{ij} = [-b_{ji}, b_{ij}]$. In this paper we will use both notations. As a special case we mention the interval constraint $0 \leq x_i - x_j \leq \infty$ specifying that $x_j$ has to occur before $x_i$. Let us now present a simplified example of a STN in the turnaround domain.

**Example 2.1** Suppose we have a flight with an on and off-block time of 13:00 and 15:30, respectively. Therefore, all required groundservices like fuelling, (de)boarding and cleaning have to be done between 13:00 and 15:30. It is required that deboarding has to start within 15 minutes after on-block and it takes at least 10 and at most 20 minutes. Fuelling can only start if deboarding has ended. Finally, we know that fuelling takes at least 20 and at most 40 minutes and has to be completed 30 minutes before off-block. We can model these data as a STN now. We use the following nodes:

\[
\begin{align*}
x_0 &= \text{temporal referential point (0 = 12:00.0)} & x_4 &= \text{begin time fuelling} \\
x_1 &= \text{on block time} & x_5 &= \text{end time fuelling} \\
x_2 &= \text{begin time deboarding} & x_3 &= \text{end time deboarding} \\
x_6 &= \text{off-block time}
\end{align*}
\]

And the following constraints:

\[
\begin{align*}
60 &\leq x_1 - x_0 \leq 60 & \text{on-block exactly 60 minutes after 12:00:00} \\
150 &\leq x_6 - x_1 \leq 150 & \text{the time between on- and off-block is 150 minutes} \\
0 &\leq x_2 - x_1 \leq 15 & \text{deboarding has to start within 15 minutes after arrival} \\
10 &\leq x_3 - x_2 \leq 20 & \text{duration of deboarding is between 10 and 20 minutes} \\
20 &\leq x_5 - x_4 \leq 40 & \text{fuelling takes 20 to 40 minutes} \\
0 &\leq x_4 - x_3 \leq \infty & \text{fuelling starts after deboarding has ended} \\
30 &\leq x_6 - x_5 \leq \infty & \text{fuelling has to be completed at least 30 minutes before off-block}
\end{align*}
\]

Figure 1 contains a graphical representation (a Simple Temporal Network) derived from this STP.

![Figure 1: A Simple Temporal Network](image)

The solution of a STN is a specification of suitable values for time variables:

**Definition 2.2** [5] A solution for a STN $S = (X, C)$ is a complete collection of assignments $\{x_0 = 0, x_1 = w_1, \ldots, x_n = w_n\}$, with each $w_i \in \mathbb{Z}$, in such a way that all constraints are satisfied.

If such a solution exist, we say that the STN $S$ is consistent, else it is said to be inconsistent. There is an efficient algorithm to check (in)consistency of a STN based on the direct labelled graph representation $G_S = (N_X, E_C, l)$ of $S = (X, C)$, where $N_X$ is the set of nodes $n_i$ representing the time points $x_i$, $E_C$ is
the set of directed arcs, where \(e = (n_i, n_j) \in E\) has label \(l(e) = b_{ij}\) whenever \(c_{ij} : x_j - x_i \leq b_{ij}\) occurs in \(C\). If the labels \(l(e)\) are interpreted as distances, the well-known \(O(n^3)\) Floyd-Warshall All-Pair-Shortest-Path (APSP)\([2]\) algorithm can be used to determine the distance \(d(i,j)\) between all nodes \(n_i\) and \(n_j \in N_X\).

The inconsistency of \(S\) now can be decided by checking whether \(d(i,i) \geq 0\) holds for all nodes \(n_i \in N_X\).

The graph \(G^2_S\) obtained by applying the APSP algorithm to the STN \(G_S\) containing all shortest distances between the nodes in \(N_X\) and therefore specifying the tightest constraints between the time variables in \(X\) is called the \(d\)-graph associated with \(S\). Note that the \(d\)-graph is a complete graph and contains for every node pair one edge with the distance between them as label.

### 2.2 The Temporal Decoupling Method

We modelled the collection of turnaround plans using as a STN. This STN contains the specifications of all activities to be performed by the different agents (service providers) whilst on-block. In order to provide a solution for the complete turnaround process we need to specify the values of all begin and endpoints of these activities. To ensure a valid solution this seems to require either a centralized solution process or a rather elaborate coordination process requiring negotiation between the different agents involved. There is, however, another possibility. This requires a modification of the original STN \(S\) such that

- The STN \(S\) is split into \(k\) sub STNs according to the activities belonging to the \(k\) participating agents (service providers)
- Each of the agents is allowed to solve its own sub STN \(S_i\) and to specify its own solution \(s_i\) for it.
- whatever solutions \(s_i\) are chosen by the agents, their merge \(s = (s_1, s_2, \ldots, s_k)\) always constitutes a valid solution to \(S\)

This so-called temporal decoupling method specified by Hunsberger \([5]\) can be defined in a more formal way as follows:

**Definition 2.3 (z-partition)**

Let \(X, X_X\) and \(X_Y\) be collections of timepoint variables. We say that \(X_X\) and \(X_Y\), z-partition \(X\) if:

- \(X_X \cap X_Y = \{z\}\), where \(z = x_0 = 0\)
- \(X_X \cup X_Y = X\)

**Definition 2.4 (Temporal Decoupling)**

A temporal decoupling of the STN \(S = \langle X, C \rangle\) is a pair of STNs \(S_X = \langle X_X, C_X \rangle\) and \(S_Y = \langle X_Y, C_Y \rangle\) such that:

- \(S_X\) and \(S_Y\) are consistent,
- \(X_X\) en \(X_Y\) z-partition \(X\), and
- the merging of solutions for \(S_X\) and \(S_Y\) always is a solution for \(S\).

The definition of the z-partition and the temporal decoupling for more than two sets is analogous to that for 2 sets. In our application we choose the z-partition in such a way that all timepoints belonging to activities of one service provider occur in one z-partition. We will now give an example of z-partitions and removing the dependencies.

**Example 2.2** Let us assume we have a fuelling company and a boarding company who have agreed to perform two tasks, \(q_1\) and \(q_2\) with the following temporal constraints: \(3:00 \leq q_1 \leq 5:00, 3:00 \leq q_2 \leq 5:00\) (on and off-block time) and \(q_1 \leq q_2\). The fuelling company performs \(q_1\) and the boarding company performs \(q_2\). Assume that performing task \(q_1\) and \(q_2\) both cost 15 minutes. Both companies don’t want to coordinate their activities. Therefore we will choose the z-partitions in such a way that the fuelling and boarding company are both in different z-partitions. Note that both z-partitions only have \(x_0\) in common (See Figure 2). To remove the dependency between the companies we have to satisfy the direct link \(q_1 \leq q_2\) between them. This is accomplished by changing the constraint \(3:00 \leq q_1 \leq 5:00\) into \(3:00 \leq q_1 \leq 4:00\) and the constraint \(3:00 \leq q_2 \leq 5:00\) into \(4:00 \leq q_2 \leq 5:00\). The effect of this change is that the direct edge between the companies is always satisfied (the path between them through \(x_0\) is dominating this edge connecting them directly) and can be removed. As a consequence, the time interval in which both companies have to perform their task is smaller, but they can perform their tasks independently from one another.
In [5, 4] an algorithm is described that can solve TDPs.

### 3 Determining the number of resources needed

With the help of the decoupling algorithm an independently schedulable temporal plan per agent (service provider) can be achieved. In general such a schedule will also include the number of resources needed in order to complete all the activities specified in the temporal plan of the service provider. For example, agents such as the fuelling or catering company need vehicles as resources to service an aircraft. It is possible for one resource to service multiple aircrafts if the activities take place after each other and there is sufficient time between them to travel. Hence, we need to take into account the (travelling) time between the activities to be performed by each agent. To give an example, let us consider the temporal plan of the fuelling agent. In such a plan the Earliest Starting Time $EST(a)$, Latest Starting Time $LST(a)$, Earliest End Time $EET(a)$ and the Latest End Time $LET(a)$ are specified per activity $a$. Table 1 shows an example in which for each flight it is indicated at what time the fuelling agent must start (earliest) and when the agent must end (latest).

In order to determine the number of fuelling vehicles for this agent $A$, we model the set of activities $A_c^A$ specified in the decoupled temporal plans as the set of nodes $V$ in a graph $G_A = (V, E)$. There is a directed edge between two activities $a_1, a_2 \in V$ if it is possible to accomplish activity $a_2$ after $a_1$ with the same resource. Here, we can use two methods to determine whether $a_2$ is serviceable after $a_1$: the minimum and the maximum method.

- minimum method: $EET(a_1) + \text{distance}(a_1, a_2) \leq LST(a_2)$
- maximum method: $LET(a_1) + \text{distance}(a_1, a_2) \leq EST(a_2)$

Here, $\text{distance}(a_i, a_j)$ is a table with travel times (distances) between the gate for $a_i$ and the gate where $a_j$ has to be performed. In the next example we show how to construct such a graph using the maximum method.

**Example 3.1** We construct the graph $G_A = (V, E)$ for the fuelling agent. First, we need its decoupled plan with a list of activities with all the flights and the $EST$, $LST$, $EET$ and $LET$ for the fuelling service for these flights. Table 1 shows a part of this list. We also need a table with the travel times (distances) between gates. We check if for every possible pair of activities $a_1$ and $a_2$ the equation $LET(a_1) + \text{distance}(a_1, a_2) \leq EST(a_2)$ holds. If so, we add a directed edge $(a_i, a_j)$ to $G_A$. If not, no edge is added. For example, if we check for flight KL1857 and KL1577, we have $13:28:31 + 00:02:00 > 12:25:23$. It follows that it is not possible to service these two flights after each other and consequently no edge is added. Applying this method to all pairs results in Figure 3.

In Figure 3 there is a path from flight KL1857 via KL8437 to KL0435. This means that only one resource is needed to service these three flights. In order to determine the minimum number of resources needed, we now have to find out how many paths we need if we want to cover all nodes exactly once. In the literature, this problem is called the Minimal Node Disjoint Path Cover problem (e.g., [6]). In general, this problem is intractable, because the decision variant (Node Disjoint Path Cover) is a special case of the NP-complete

---

1. This algorithm uses a so-called greediness factor which is used to select the balance between flexibility of the solution and computation time.
Table 1: Decoupled plans for fuelling service

<table>
<thead>
<tr>
<th>Call sign</th>
<th>Gate</th>
<th>EST</th>
<th>LST</th>
<th>EET</th>
<th>LET</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL1857</td>
<td>C06</td>
<td>12:00:00</td>
<td>12:00:00</td>
<td>12:12:00</td>
<td>13:28:31</td>
</tr>
<tr>
<td>KL1013</td>
<td>B13</td>
<td>12:10:00</td>
<td>12:10:00</td>
<td>12:22:00</td>
<td>13:39:37</td>
</tr>
</tbody>
</table>

Figure 3: Graph $G_A$ for the fuelling company created with maximum method.

HAMPAD problem (Covering all nodes with one path). In our case, however, the graph $G_A$ is acyclic\(^2\), which, as we will show, implies that the problem can be solved in polynomial time.

To solve this problem, we will show here that the minimum number of paths can be determined by reducing the problem to a polynomial solvable Maximum Flow problem\(^7\). First, we will construct a flow graph $G_F$ from the graph $G_A$ presented in Figure 3. Second, we will prove that a solution of this Maximum Flow problem can be easily converted into a solution of our Minimal Node Disjoint path Cover problem, i.e., the minimum needed capacity problem.

First, we construct from the directed graph $G_A = (V, E)$, created with maximal or minimal method, a new (flow) graph $G' = (V', E')$ as follows. Let $V = \{x_1, x_2, ..., x_n\}$.

- $V' = \{s, x_1, x_2, ..., x_n\} \cup \{y_1, y_2, ..., y_n, t\}$, where $s$ (the source) and $t$ (the sink) are two nodes not occurring in $V$ and for every $x_i \in V$, $y_i$ is a new node in $V'$;

- $E' = \{(s, x_i) : x_i \in V\} \cup \{(y_i, t) : x_i \in V\} \cup \{(x_i, y_j) : (x_i, x_j) \in E\}$.

Performing this transformation on the graph $G_A$, we get the flow graph $G_F$ depicted in Figure 4. The top row is called the X-row with the $x_i$ nodes and the bottom row of nodes is the Y-row of copies $y_i$. We give each edge $e \in E'$ a capacity of 1. Executing a Maximal Flow algorithm on $G_F$ gives the maximum amount of flow in the graph and the edges through which it flows (See Figure 4, the edges in the maximal flow are marked bold). The flow $f$ is a maximum flow with value 6. We now show that, in general, the value of the maximal flow in $G_F$ determines the solution specifying the minimum amount of resources needed (i.e. the solution to the minimum disjoint path cover problem).

**Proposition 3.1** Given a graph $G = (V, E)$ and the constructed flow graph $G' = (V', E')$, let $f$ be the value of the maximal flow of $G'$. Then the minimum number of paths needed for a disjoint path cover of $G$ is $|V| - f$.

**Proof** See Appendix.

\(^2\)Note that the graph is always transitive: if there exists an edge between node $a$ and $b$ then, according to the maximal method, $a_{let} + \text{distance}(a,b) \leq b_{let}$ holds. If there also exists an edge $(b,c)$ then $b_{let} + \text{distance}(b,c) \leq c_{let}$ also holds. By definition $b_{let} \leq a_{let}$, from which follows that $b_{let} + \text{distance}(b,c) \leq c_{let}$ and thus $a_{let} + \text{distance}(a,b) + \text{distance}(b,c) \leq c_{let}$ and $a_{let} + \text{distance}(a,b) \leq c_{let}$. This transitivity of the graph ensures that the requirement that each flight is serviced (fuelled) exactly once doesn’t restrict the solution set.
When we apply the Maximum Flow algorithm to our example in Figure 4 we find a maximum flow of 6. Thus, the minimum number of resources needed to carry out all activities is 8 (The number of activities - the maximum flow).

Note that in addition to the value of the pathcover the algorithm also determines which flights can be serviced by the same resource. Each edge with positive flow is a part of a path. In Figure 4 these edges are marked bold. Constructing the paths is now easy. We take for each edge \((x_i, y_j)\) with flow the edge \((v_i, v_j)\) in \(G\) and add these to the path cover. Succeeding nodes are on the same path. Nodes not belonging to a path belong to their own path with length zero. In Figure 3 the paths are the following ones:

- Paths length 1:
  \{(KL1857, KL3411)\}, \{(KL1013, KL8437)\}, \{(KL1667, KL4103)\}, \{(KL1577, KL1725)\}, \{(KL8004, KL1795)\}
  and \{(KL1113, KL0435)\}

- Paths length 0:
  KL0713 and KL8114

In Figure 5 the resources needed are plotted against time. The figure demonstrates that the maximum capacity of 8 vehicles is only needed between 13:00 and 16:00 - not for the complete time period.
4 Discussion

In this paper we discussed a multi-agent approach to coordination with an application to the turnaround process on airports. In particular, we discussed a temporal decoupling method that could be used to offer service providers at an airport a stand alone specification of their activities which enables them to plan/schedule their activities independently of the planning activities of other agents. Given this decoupling of turnaround plans, we presented an algorithm to determine the number of resources needed to execute all the activities in the decoupled plan, based on the (earliest en latest) start and end times and the travelling time between activities. The time complexity of this algorithm is rather modest: it is bounded above by the complexity of the Maximum Flow algorithm and lower bounded by $O(n^2)$.

An advantage of this algorithm is that it not only gives the amount of needed resources, but also where and when these should operate. For example, figure 5 shows that the minimum capacity of 8 is only needed for three hours on the whole day and that the rest of the time the aircrafts can be serviced with less resources. It is possible to have different path covers with the same maximum value. For agents these path covers can be more or less efficient, because the total distance driven by resources could be lower in another path cover. It would be an improvement to use a weighted Maximum Flow algorithm, where the weight on an edge is the distance between the two activities. Further research can also be done on optimizing the selection and length of paths in the path cover in order to divide the load between individual resources.

In this paper we made a few assumptions in order to reach an efficient solution. Our methods for determining the edges use the duration of an activity as one value (20 or 30 minutes) instead of an interval (20 to 30 minutes). We did provide maximal and minimal methods which give the lower and upper bound of the resources needed. We also assumed that an aircraft needs the same kind of service no matter at what gate it is serviced. For example, each aircraft in our algorithm needs to be fuelled with a fuelling vehicle, while some airports have fuel depots on certain gates and no vehicle is necessary. Another assumption is that resources are always able to service. For example, in our algorithm a fuelling vehicle is able to service an unlimited amount of aircrafts without refuelling itself. An important extension to the algorithm would be the adding of these specific constraints which give a much more realistic view on resource movement.

References


---

In [1] the fastest algorithms available up till now are presented.
A Proof of Proposition

**Proposition A.1** Given a graph $G = (V, E)$ and the constructed flow graph $G' = (V', E')$, let $f$ be the value of the maximal flow of $G'$. Then the minimum number of paths needed for a disjoint path cover of $G$ is $|V| - f$.

**Proof** We show that $G'$ has a flow with value $|V| - k$ if and only if $k$ is set of disjoint paths covering $V$. From this correspondence it follows immediately that maximal flows correspond to minimum disjoint path covers.

$(\Leftarrow)$ Given graph $G$ with a path cover consisting of $k$ disjoint paths, there are $k$ nodes as the starting point of a covering path and $|V| - k$ which are not. Because the paths are node disjoint, each of these $|V| - k$ nodes has a unique predecessor. We construct a flowgraph $G'$ according to the method described earlier. For each node $v_{i+1}$ with predecessor $v_i$ there is an edge in $G'$ between $x_i$ from the X-row and $y_{i+1}$ from the Y-row over which one unit of flow can be pushed. Because each node has a unique predecessor, the capacity constraint holds because each node $x_i$ has at most one outgoing flow and each $y_{i+1}$ has at most one incoming flow. So $G'$ has as flow of $|V| - k$.

$(\Rightarrow)$ Consider flow of size $|V| - k$ in the graph $G'$ constructed from graph $G = (V, E)$. It is not difficult to see that this implies that there are $k$ disjoint paths from the source $s$ to the sink $t$. Hence, there are $k$ disjoint nodes (capacity is 1) in the X-row from which 1 unit of flow flows to a unique node in the Y-row. From this follows that there are $k$ nodes in $G$ that are a successor of some node on a flow path, and $|V| - k$ that are not. Clearly, these $|V| - k$ nodes are the starting point of a covering path. Therefore, there are $|V| - k$ paths in the path cover for $G$. 