MISTA 2011

A more realistic approach for airport ground movement optimisation with stand holding

Jason A. D. Atkin · Edmund K. Burke · Stefan Ravizza

Abstract Airports are already facing several challenges and can expect to face more in the future. Despite the requirements to handle ever increasing numbers of aircraft, they also have to meet environmental targets and regulations. Although physical expansion of the airport could sometimes help, this is rarely possible in practice. The complexity increases the closer an airport has to work to their maximal possible capacity. The complexity of the problem means that advanced decision support systems are needed to guarantee efficient airside airport operations and to mitigate the environmental impact. This research considers the important problem of getting aircraft from source to destination locations (usually either runways or gates/stands) in as efficient a manner as possible, in terms of time or fuel burn. A new sequential graph-based algorithm is introduced for this important part of the airside operations at an airport - usually named the ground movement problem. This algorithm, embedded in a wider operational system, has several advantages over previous approaches for increasing the realism of the modelling and it utilises a recently developed approach to more accurately estimate taxi speeds. Importantly, the taxi time prediction and the ground movement model use the same directed graph representation of the airport. For the experiments in this paper, the algorithm has been configured to absorb as much of the waiting time as possible for departures at the gate/stand, to reduce the fuel burn and the environmental impact. Analysis with data from a European hub airport shows very promising results and gives an indication of both the performance of the system (in comparison to a lower bound on the taxi time) and the limits to the amount of waiting time which could possibly be absorbed as stand hold (without the engines running).

1 Introduction

European airports face several challenges in the 21st century, including the capacity challenge (with demands for air travel still increasing year on year) and the environmental challenge [1]. To avoid forming huge bottlenecks in the air transportation

Jason A. D. Atkin, Edmund K. Burke and Stefan Ravizza (corresponding author)
School of Computer Science, University of Nottingham
Jubilee Campus, Nottingham, NG8 1BB, UK
E-mail: jaa@cs.nott.ac.uk, ekb@cs.nott.ac.uk, smr@cs.nott.ac.uk

system, airports have either to be enlarged, or (since enlargement is not possible in most cases), to utilise the existing resources as efficiently as possible. In addition, the increasing focus upon environmental issues is likely to further grow over time.

As airports work closer to their maximum capacity, airside airport operations become much harder to deal with. As a result, decision support systems have to be increasingly advanced and need to both integrate different airside airport operations with each other and also model each process increasingly realistically.

From an optimisation point of view, ground movement of aircraft can be considered to be one of the most important airside operations at an airport [6], since it links several other problems together, such as the runway sequencing problems for arrivals and/or departures [4], the stand holding problem [2,3] and the gate assignment problem [11]. For a comprehensive literature review of ground movement research and the integration with other operations, we point the interested reader to a recently published review [6].

This paper presents a decision support framework for environmentally friendly ground movement and promising experimental results which utilise more realistic taxi time predictions for a European hub airport. A framework is described for integrating a graph-based sequential movement algorithm into a larger decision support system which can also consider the runway sequencing problem and the stand holding problem. A statistical approach has been used to more accurately estimate taxi times for aircraft than a standard lookup table may allow. This utilises the same graph which is used for the ground movement model. This integrated approach allows the effects of ground plan changes to be modelled more accurately, changing both taxi speed predictions and routing information. In addition, several concepts have been included in the model which allow airport layouts to be modelled in a more realistic manner, such as restricting certain taxiways to be used only by certain aircraft and coping with the required separations between aircraft. Finally, the absorption of delay at the stand, prior to starting the engines, has been considered. This reduces the waiting times at the runway and is further extending previous stand holding ideas [2,3,8]. The maximal potential benefits of such a system have been quantified.

Section 2 provides a description of the airport ground movement problem and how it can be embedded into the larger combined sequencing/routing/stand holding framework. The test dataset from the airport is then presented in Section 3 together with the statistical estimation of taxi times. Following this, the sequential ground movement algorithm which has been developed, and was utilised for these experiments, is detailed in Section 4. The results of the application of the algorithm to the test dataset are then shown in Section 5; before the paper ends with some conclusions in Section 6.

2 Problem description

The links between the ground movement problem and runway sequencing are considered first in this section, before the ground movement problem itself is discussed in more detail. The section ends with a consideration of the stand holding benefits which can result from the appropriate solution of the ground movement problem.

2.1 The links with runway sequencing

Atkin et al. [6] highlighted the importance of integrating the ground movement problem with other airside airport operations, such as the problems of finding good departure and arrival sequences. Supporting controllers in these tasks is a challenge, especially when departures and arrivals have common restrictions and interactions due to the airport layout. For this paper, we assume that the runway sequencing and ground movement problems are solved as two distinct stages. The integrated (departures and arrivals) runway sequencing problem is solved in a first stage, then the consequent landing and take-off times are used in the second stage within the consideration of the ground movement problem. Thus, the wheels-on time at the runway (for arrivals) and the wheel-off time at the runway (for departures) are both assumed to be fixed within the ground movement problem. Later research will analyse the benefits of providing a feedback loop from the ground movement problem to the integrated runway sequencing problem and of closer integration between the two problems.

2.2 Problem description of the ground movement problem

This paper considers ground movement at an airport. The ground movement problem is a combined routing and scheduling problem. It involves guiding aircraft on the surface of an airport to their destinations in a timely manner, where the goal is to reduce the overall travel time and to enable the target take-off times at the runway to be met. It is important, for reasons of safety that two aircraft never conflict with each other throughout the ground movement process.

In the model which is considered in this paper, the route of the aircraft is not pre-determined, allowing greater flexibility for solutions; however, the utilised solution method provides the possibility to restrict certain aircraft to specific taxiways and/or to avoid routes which involve tight turns. The airport layout is represented as a directed graph, where the edges represent the taxiways and the vertices represent the junctions or intermediate points. Aircraft are considered to occupy edges, and conflicts are avoided by preventing any two aircraft from using the same edge simultaneously.

As we will discuss in the next section, the aircraft movement speed depends upon various factors and, in order to provide a more realistic ground movement decision support system, it is important to take this variation into account. The times at the runway are assumed to be fixed for departures as well as for arrivals. The sequential approach to ground movement will then minimise the taxi time for each individual aircraft given the movement of the aircraft which have already been routed. Hence, the approach will attempt to absorb as much of the waiting time as possible at the gate/stand. Consequently, the aircraft can start its engines as late as possible, reducing fuel burn and environmental impact. Thus, the solution method can be considered to be not only reducing the ground movement time, but also solving the stand holding problem [2, 3, 8].

3 Analysed case: Zurich Airport

This analysis utilised data from Zurich Airport (ZRH), which is the largest airport in Switzerland and a hub airport for Swiss International Air Lines. The airport has three

runways, named 10/28, 14/32 and 16/34, according to their direction of operation, with two runways intersecting each other (see Figure 1). The considered data included information about the airport layout, the positions of stand and runway entrance and exit points and the layouts of all of the taxiways as well as the real timings for the various aircraft. This information was used to develop a taxi time prediction function, as discussed below, to improve the accuracy of the taxi time predictions which are used in ground movement. We had access to data for an entire day's operations for the 19th of October 2007. No extraordinary occurrences took place and there were 679 movements in total (337 arrivals and 342 departures).

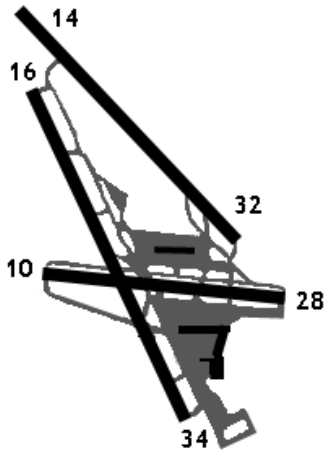

Fig. 1 Sketch of Zurich Airport (ZRH)

Ground movement models need taxi time prediction, but sufficiently accurate values are rarely available. In previous research, average taxi speeds have been assumed to be influenced only by the aircraft type. However, other research has shown that the aircraft type is not the most influencing factor upon taxi speeds [5,13,17]. Comparisons between ground movement tool results and the status quo at airports have previously been hard to analyse, due to the need for accurate taxi speed data, since historic data usually includes the effects of any delays or re-routing due to routing/timing conflicts between aircraft. Thus, the effects of taxi time variability and the benefits from the ground movement decision support system were often intermingled.

An approach to more accurately predict taxi times for aircraft or, equivalently, their average speeds, was proposed in Atkin et al. [5]. Multiple regression was used to estimate a function which could more accurately predict the taxi speed and identify the factors which were most related to taxi speeds, such as total distance travelled, total turning angle and the number of other aircraft of different types which were on the airport's surface. The aim was to then eliminate the effects of factors which represented the actual amount of traffic at the airport, with the goal being to predict the taxi times for unimpeded aircraft. These could be used in a more advanced ground movement decision support system, such as the one described in this paper, providing the opportunity to compare results with the way in which an airport was actually operating. Such a system will itself model the effects of the interaction between aircraft (so

these should not be included in its taxi speed data). The resulting taxi time prediction functions were utilised to predict the taxi times which were employed in this paper.

Depending upon the terminal and the operating mode (which runways are in use), runway crossings may be necessary during the taxi process. We plan to integrate these effects into the combined ground movement and sequencing model later, but for the moment they are included in the statistical model for taxi speeds.

4 Ground movement decision support system

In the routing approach described in this paper, the aircraft are routed sequentially. When an aircraft is ready, it has to be routed respecting previous reservations by other aircraft using the taxiways. The routes which have been previously calculated for other aircraft do not change as new aircraft are taken into consideration. Thus to acknowledge the difficulty and time costs associated with communicating changes to pilots and reducing the quantity of communication needed between the surface controllers and the pilots. The objective for each of the sequential routings is to find the routing with minimal taxi time among all remaining conflict-free routings.

The approach described here is based on research by Gawrilow et al. [12] and the PhD thesis of Stenzel [21]. Ravizza modified the approach for his Master's dissertation [18] to label the vertices instead of the edges to simplify their interpretations. The original aim of this approach was to control automated guided vehicles in container terminals in harbours or in storage areas but is here applied instead to routing aircraft. The approach has then been further modified for this work, for instance by allowing the approach to work backwards to meet a specified end time rather than starting time. The resulting algorithm is described in this section.

A directed graph model of the airport is used by the algorithm. The so called Quickest Path Problem with Time Windows (QPPTW) algorithm is a generalized vertex-based label-setting algorithm based on Dijkstra's algorithm and can sequentially route aircraft on the airport surface. In contrast to many other ground movement support systems which use time discretisation [7, 15, 16, 19], no such discretisation is used in this approach. It has similarities to the recently published work by Lesire [14], which used a sequential A* algorithm, but it provides a better coverage of the solution space, potentially allowing it to find better solutions within comparable execution times - these being short enough for it to be appropriate for real-time decision making.

The algorithm which is used in this paper also provides the possibility to define which edges in the graph are in conflict with each other and cannot be used simultaneously. In addition, for each edge incident to a vertex, the set of valid outgoing edges can be manually defined if desired, or can depend upon information about the aircraft. This enables the decision support system to forbid aircraft from making narrow turns or prevent aircraft from using taxiways for which they are too large. Together, these features enable the approach to more realistically model the airport surface while leaving the routing task itself to the algorithm.

The preprocessing of the algorithm is explained in Section 4.1, before introducing the QPPTW algorithm. The section ends with the discussion about buffer times and the sequencing of the routed aircraft.

4.1 Ground plan preprocessing

It is important to maintain separations between aircraft on the ground. The concept of conflicting edges is introduced here for this reason, so that no two conflicting edges can be occupied simultaneously. The conflicting edges are determined in a preprocessing stage. For this research, we used an approach which assumes straight connecting lines between vertices since this requires less time in the preprocessing stage and is adequate for the directed graph model which has been used in this research, where aircraft drive in an almost straight line between vertices. In this approach, two edges conflict with each other if they are located closer together than a given threshold distance. Edges in the graph, together with their embedding in the airport plan, are here named segments. To find the minimal Euclidian distance between two segments, the algorithm performs two processing steps. Firstly, it verifies whether the edges are intersecting, then, if they are disjoint, the distance between each end point of one segment and the closest point on the other segment is calculated. The minimum over these four distances corresponds to the minimal distance between the two segments.

4.2 Variable definitions

Definitions of the variables and data structures which are used in the model are given in Table 1.

Table 1 Table of definitions

Variable	Explanation
$confl(e)$	The set of edges which conflicts with edge $e \in E$
$F_e = [a_e, b_e]$	A time-window on edge $e \in E$
$\mathcal{F}(e)$	The sorted set of all the time-windows on edge $e \in E$
$G = (V, E)$	The directed graph representing the airport layout, with vertices $v \in V$ and edges $e \in E$
H	The Fibonacci heap storing the added labels
$I_L = [a_L, b_L]$	The time interval used in a label L
$L = (v_L, I_L, pred_L)$	A label on vertex $v_L \in V$ with time interval I_L and predecessor label $pred_L$
$\mathcal{L}(v)$	The set of all of the labels at vertex $v \in V$
R	A conflict-free route that is being generated
$T = (s, t, time)$	A taxi request to route, from source $s \in V$ at time $time$ to target $t \in V$
w_e	The weight (necessary taxi time) of edge $e \in E$

4.3 Key concepts

The QPPTW algorithm with its expansion steps works in a similar way to Dijkstra's algorithm [9,10]. However, in the QPPTW algorithm, a label can be expanded several times due to the different time-windows and a concept of dominance is needed in order

to guarantee a polynomial solution time. It is necessary to define some of the concepts upon which the approach is based. Firstly, the algorithm needs information about the times that each part of the taxiway (edge) is free and this is provided in the form of a sorted set of free time-windows for each edge:

Definition: Set of sorted time-windows
The set $\mathcal{F}(e)$ contains the sorted set of time intervals $F_e^j = [a_e^j, b_e^j]$ which specify the times when the edge e can be used for a new route. This will exclude the times when e, or an edge which conflicts with e, are in use by previously routed aircraft. These are inputs to the routing algorithm for each aircraft.

The use of labels is the essential concept of the QPPTW algorithm.

Definition: Label
A label $L = (v_L, I_L, pred_L)$ *specifies the time period $I_L = [a_L, b_L]$ within which the current aircraft could reach vertex v_L. It includes a reference to the previous label on the route, $pred_L$, and thus implicitly represents a route (with edge traversal timings) from a source vertex to the specified vertex v_L. These labels are generated as the routing algorithm progresses, together specifying the (undominated) time periods (from time a_L to time b_L) when the current aircraft could reach vertex v_L.*

An ordering relation is defined over the intervals of the labels to allow the definitions of dominance.

Definition: Dominance
A label L dominates *a label L' on vertex $v_L = v_{L'}$ if and only if $I_{L'} \subseteq I_L$ (and there are identical route restrictions on the outgoing edges), which implies $a_L \leq a_{L'}$ and $b_L \geq b_{L'}$.*

Once the routing has been performed by the QPPTW algorithm, the time-windows are readjusted (as discussed in Section 4.6) before the QPPTW algorithm is reapplied to route the next aircraft.

4.4 QPPTW algorithm

The input of the QPPTW algorithm contains the graph $G = (V, E)$ with its weight function w_e, which corresponds to the taxi times for each edge, estimated using the taxi time estimation method which was described in Section 3. The sorted set of available time-windows $\mathcal{F}(e)$ also has to be provided for each edge e, specifying when the edge is available. A taxi request $T_i = (s_i, t_i, time_i)$ for aircraft i is then a conflict-free route R from the vertices s_i to t_i with minimal taxi time (w.r.t. w_e) that respects the given time-windows.

The pseudocode of the QPPTW algorithm is shown in Algorithm 1 and is a close variant of the QPPTW algorithm described by Stenzel in [21]. The main difference is that we allocate the labels to vertices, which helps both to model the process more realistically and to more easily understand the algorithm, since it distinguishes between the use of the labels at the vertices and the input time-windows at the edges.

Lines 1 and 2 of Algorithm 1 involve the initialization of the Fibonacci heap and the references to the Fibonacci heap which are stored at each vertex. The use of Fibonacci

Algorithm 1: Quickest Path Problem with Time Windows (QPPTW)

Input: Graph $G = (V, E)$ with weights w_e for all $e \in E$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$, a taxi request $T_i = (s_i, t_i, time_i)$ with the source vertex $s_i \in V$, the target vertex $t_i \in V$ and the start time $time_i$.

Output: Conflict-free route R from s_i to t_i with minimal taxi time that starts at the earliest at time $time_i$, respects the given time-windows $\mathcal{F}(e)$ or returns the message that no such route exists.

```
1  Let H = ∅
2  Let L(v) = ∅   ∀v ∈ V
3  Create new label L such that L = (s_i, [time_i, ∞), nil)
4  Insert L into heap H with key time_i
5  Insert L into set L(s_i)
6  while H ≠ ∅ do
7      Let L = H.getMin(), where L = (v_L, I_L, pred_L) and I_L = [a_L, b_L]
8      if v_L = t_i then
9          Reconstruct the route R from s_i to t_i by working backwards from L
10         return the route R
11     forall the outgoing edges e_L of v_L do
12         foreach F^j_{e_L} ∈ F(e_L), where F^j_{e_L} = [a^j_{e_L}, b^j_{e_L}], in increasing order of a^j_{e_L} do
13             /* Expand labels for edges where time intervals overlap */
14             if a^j_{e_L} > b_L then
15                 goto 11 /* consider the next outgoing edge */
16             if b^j_{e_L} < a_L then
17                 goto 12 /* consider the next time-window */
18             Let time_in = max(a_L, a^j_{e_L})    /* a^j_{e_L} > a_L ⇒ waiting */
19             Let time_out = time_in + w_{e_L}
20             if time_out ≤ b^j_{e_L} then
21                 Let u = head(e_L)
22                 Let L' = (u, [time_out, b^j_{e_L}], L)
23                 /* dominance check*/
24                 foreach L̂ ∈ L(u) do
25                     if L̂ dominates L' then
26                         goto 12 /* next time-window */
27                     if L' dominates L̂ then
28                         Remove L̂ from H
29                         Remove L̂ from L(u)
30                 Insert L' into heap H with key a_{L'}
31                 Insert L' into set L(u)
32 Output: there is no s_i-t_i route
```

heaps for this algorithm has the same beneficial effect upon the execution time as it does for Dijkstra's algorithm. The starting label is generated for the source s_i in line 3 and is then inserted into the Fibonacci heap, which is sorted with respect to the earliest possible arrival time (key). A reference is maintained to this label using the $\mathcal{L}(s_i)$ set for each vertex. These references are used as a look-up by the dominance check in

lines 23-29, where the algorithm needs fast access to all of the labels associated with a particular vertex.

In each while-loop, the algorithm checks whether the Fibonacci heap still contains elements. If this is not the case, there is no route which can be enlarged and, therefore, no route from s_i to t_i, starting at $time_i$, exists (line 32). If the Fibonacci heap still contains elements, the algorithm takes a minimal element with respect to the key (line 7), checks whether this label already represents a route to the target t_i (line 8-10) or, otherwise, tries to expand the associated route.

The route can usually continue along a number of different outgoing edges from any vertex and can potentially use different time-windows on each edge (line 11 and 12). In order to use an edge there must be a time-window available with an overlapping time interval, as expressed by the conditions in line 14 and 16. The earliest possible point in time that edge e_L can be left is identified (line 18 and 19) and the expansion step is executed. When the condition stated in line 20 is true, a new label will be generated (line 21 and 22). Different cases are possible at this stage. Firstly, the new label may dominate another label (line 27), in which case the dominated label will be erased (line 28 and 29). Secondly, the new label may be dominated by an older one (line 25), in which case it is not necessary to take this label into account (line 26). The while-loop is executed as long as there is a route which can be expanded. If a route R to the target t_i has been found, the route can be generated by working backwards through the set of labels (line 9) using the references, $pred_L$, to the previous labels.

The generalized vertex-based Dijkstra's algorithm which has been presented here is a close variant of that given by Stenzle in [21] and solves the problem in polynomial time (polynomial in the number of time-windows). The proof can be found in [21], where it is presented for the edge-based algorithm.

4.5 Modifications to the QPPTW algorithm for airport ground movement

Algorithm 1 is used for arriving aircraft as described above, since their goal is to clear the runway and reach the gate/stand as quickly as possible. In our model, departing aircraft aim to reach the runway at a given time and leave the gate/stand as late as possible in order to do so. This allows for more of the waiting time to be absorbed at the gate/stand when the engines are not running. The same algorithm is used for this purpose, computing the route backwards with the end time fixed instead of the start time, with only minor changes to reverse the time-related steps. Since the algorithm logic remains unchanged, this modified algorithm has not been presented here.

To attempt to speed up the execution time of the algorithm, we applied goal-oriented search [20] to the QPPTW algorithm. Two heuristic measures were used to estimate a lower bound for the rest of the partial route: the Euclidean distance was used to measure the linear distance to the target, and the time was estimated using Dijkstra's algorithm to compute the time which would be needed ignoring any interference from other aircraft. Unfortunately, neither approach resulted in a valuable speed-up when applied to this problem. This can possibly be explained by the fact that the graph representing the airport layout is sparse (having on average only a few outgoing edges for each vertex) and routes often start on the border of the graph (see Figure 1), so the number of expansions exploring non-promising areas of the airport is relatively small.

4.6 Readjustment of the time-windows

When an aircraft has been routed, the time-windows have to be readjusted according to the edge utilisation of the adopted route R, and the edges which conflict with these. We note that it is necessary to consider edge conflicts only during this stage and not during the routing process (Algorithm 1).

Algorithm 2 presents the pseudocode for the readjustment of the time-windows. The input consists of the weighted graph $G = (V, E)$, the set of conflicting edges $confl(e)$ for all $e \in E$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$, and the route R which was found for the most recent aircraft to be routed. The output is the new sorted set of time-windows $\mathcal{F}(e)$, including the reservations of the new route R.

Algorithm 2: Readjustment of the time-windows

Input: Graph $G = (V, E)$ with weights w_e for all $e \in E$, the route R with reservations $\left[time_f^{in}, time_f^{out}\right]$ for all $f \in R$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$ and the set of conflicting edges $confl(e)$ for all $e \in E$.

Output: Sorted set of time-windows $\mathcal{F}(e)$ including the reservations of the route R

```
1  foreach f ∈ R do
2      foreach e ∈ confl(f) do
3          foreach F_e^j = [a_e^j, b_e^j] ∈ F(e) do
4              if time_f^out ≤ a_e^j then
5                  goto 2    /* time-window is too late*/
6              if time_f^in < b_e^j then
7                  /* otherwise time-window is too early */
8                  if time_f^in < a_e^j + w_e then
9                      if b_e^j − w_e < time_f^out then
10                         Remove F_e^j from F(e)
11                     else
12                         /* shorten start of time-window */
13                         F_e^j = [time_f^out, b_e^j]
14                 else
15                     if b_e^j − w_e < time_f^out then
16                         /* shorten end of time-window */
17                         F_e^j = [a_e^j, time_f^in]
18                     else
19                         /* split time-window */
20                         F_e^j = [a_e^j, time_f^in]
21                         Insert [time_f^out, b_e^j] into set F(e)
```

Basically, the algorithm determines which other edges are blocked for each edge of the route R (lines 1 and 2). All affected time-windows on these edges are adjusted (line 3-7) and four different cases then have to be considered, depending upon the relative

positions of the time-windows. The remaining time-window may be removed (line 9-10) if it becomes too short to allow an aircraft to taxi; be shortened at the start (line 11-13) or shortened at the end (line 15-17); or it could be split in two smaller windows (line 18-21).

Once a route has been allocated to an aircraft, some additional waiting times may be required on edges, beyond the time required to traverse the edge as specified by the time intervals on the labels by Algorithm 1. Time intervals on adjacent edges often overlap sufficiently that there is a choice of which edge the wait can be assigned to. In our implementation, the waiting times are forced to be as late in the corresponding part of the route as possible (apart from the initial waiting time for departures, which is maximised to maximise the stand hold). Alternative approaches could use this flexibility to select better and smoother speed profiles for the aircraft. Using a similar approach to that used in [14], the aim could be to spread the necessary waiting times for an aircraft in such a way that the speed profiles are as "engine friendly" as possible. Although the effects of such postprocessing are not studied within this paper, they are an area which we intend to investigate.

4.7 Buffer times

The solutions of the approach are conflict-free routings, but it is possible for small delays to affect the entire plan. Buffer times would allow small deviations from the taxi times to be absorbed. To achieve such buffer times the label intervals in the algorithm are lengthened in the desired direction (before or after) by a certain amount. To reflect growing uncertainties along the route, the amount of time can be made distance-dependent. Buffer times could also depend upon the expected congestion at the time, being increased when delays were expected to be more likely, although at these times the introduction of a buffer time would be more likely to reduce throughput. This is an interesting area for future study.

4.8 Initial sequencing of taxiing aircraft

Burgain et al. [8] presented the concept of using collaborative virtual queues, extending previous gate holding concepts. The idea was to limit the number of aircraft which were taxiing on the surface and add all which want to start taxiing to a virtual queue, forcing them to wait until the count allows other aircraft to pushback. One benefit of this approach for the airlines is that they can swap their own aircraft in the virtual queue according to their priority level. Based upon this concept, the aircraft are initially ordered for the sequential routing algorithm by either the expected wheel-on time on the runway or the expected pushback time at the gate/stand. However, there are potentially better sequences which could be used to sequentially route the aircraft and we intend to analyse this potential.

5 Results and discussions

This section starts with a table collating the results to ease comparison. The explanation of the results will follow. The results of the taxi time estimation which was

presented in Section 3 are then discussed. An analysis of the results from the ground movement decision support system, which was described in Section 4, is then provided and this section ends with a summary of the potential benefits of such a system.

Table 2 Summary of the results

	Total taxi time [s]	Average taxi time per aircraft [s]
Actual total taxi time	271851.6	400.4
Statistical approach		
Total taxi time estimation	266990.5	393.2
Total taxi time estimation (unimpeded)	203331.8	299.5
QPPTW algorithm		
Using unimpeded taxi time estimates	207722.5	305.9
Using impeded taxi time estimates	276927.9	407.8

The relevant results are summarised in Table 2. The first row of results shows the actual total and average taxi times for the supplied dataset, including queuing time at the runway. The taxi speed function which was developed was then applied to each aircraft, to estimate the taxi times and the results are shown in the next two rows. In the first case, the function was applied assuming the actual traffic level and we note that the difference between the predicted and actual times is less than 2%. In the second case, the traffic related components of the function were zeroed, to estimate the taxi times if there had been no delays due to other aircraft, and the difference illustrates the amount of the taxi time which was a result of such delays. The two sets of taxi times (excluding and including delays for other aircraft) were then used within the QPPTW algorithm and the total and mean resulting taxi times are shown in the table for each case. These results are analysed and explained further in the following two sections.

5.1 Analysis of taxi time estimation

The taxi speed analysis by Atkin et al. [5] was used for taxi time prediction within this research. The goal of the multiple regression model was to find the most important factors explaining the variability of the data in the real dataset. Extensive analysis was performed, seeking a function which was as practical as possible (requiring less information), easy to interpret, and maintained a high level of accuracy.

The estimation approach has two important uses. Firstly, the overall taxi time (including runway queue time) can be estimated, taking into account the other traffic at the airport. Secondly, the approach could be used to estimate the taxi time for an unimpeded aircraft. The coefficients which were found for each factor are specified in the paper by Atkin et al. [5], and the shortest route for each aircraft between the runway and gate/stand was used for the taxi distance/speed calculation. Unimpeded taxi times for aircraft can be used in our approach since the decision support tool takes the effects of the other aircraft into consideration itself, performing the necessary re-routing or delaying of aircraft.

The results from the statistical model can be seen in Table 2. The model implies that 23.8% of the taxi time was related to delays due to other aircraft, including delays in queues behind other aircraft at the runway. There would be an average saving of 93.7s per aircraft if these interactions could be eliminated. The influence of the interactions between the aircraft which lead to the waiting times is analysed in the next section.

5.2 Experimental details using the QPPTW algorithm

The framework was programmed in Java as a single-threaded application and executed on a personal computer (Intel Core 2 Duo, 3GHz, 2GB RAM). In these experiments, all aircraft were allowed to use all of the taxiways and only intersecting and adjacent edges were considered to be in conflict and were, therefore, not allowed to be used by two aircraft simultaneously. The buffer time (Section 4.7) was set to zero. Analysis of different buffer times showed that the taxi time would have been enlarged only by a linear factor of the buffer time. Similar results were also found in [18].

Extensive analysis was performed using the QPPTW algorithm to solve the ground movement problem for the layout and data from Zurich Airport. The aircraft were routed sequentially using the taxi time estimations from the statistical analysis of taxi times which was discussed in Section 3. The resulting total taxi times can be found in Table 2, where the taxi times used were those which were estimated for unimpeded aircraft (ignoring the influence of factors related to other aircraft on the surface), the average taxi time (including re-routing and waiting delays) was 305.9s per aircraft. In contrast, the average taxi time per aircraft for the impeded taxi time estimations was 407.8s, indicating that 25.0% (or 101.9s per aircraft) of the observed taxi time was due to runway holds and re-routing or queuing due to other aircraft.

The estimations of the taxi times from the statistical approach provide a lower bound for the taxi times, since they assume no re-routing delays or queuing behind other aircraft. The QPPTW algorithm is designed to predict the delays which are actually necessary due to the interactions between aircraft for the specific routings and timings which the algorithm assigns to aircraft. Comparison of the resulting taxi times from the QPPTW algorithm against the lower bound reveals an increase in the taxi time from 203331.8 to 207722.5 seconds, showing that the additional taxi times for the re-routing and waiting summed to 4390.7s over the entire day, an increase of around 2.2% in the total taxi time. The 2.2% increase over the lower bound (rather than optimal) times indicates that its use as a ground movement decision support system seems very promising for this problem.

It is also interesting to compare the approach described here against the actual performance of the airport on this particular day of operation. Data from Zurich Airport reports a total taxi time of 271851.6s. Comparison with the results for the QPPTW algorithm with unimpeded taxi time estimation highlights savings of about 23.6% or an average of 94.5s per aircraft. Obviously, this only indicates an upper bound for the potential savings, since the real times will include some slack time for the departures at the runway to ensure a high runway throughput.

The solution time to solve the entire day of operation with 679 aircraft was 11609ms. This correlates to an average solution time of 17ms per aircraft, which supports the potential use of the algorithm in an online decision support system. No infeasible solution occurred within any of the executions of the simulation.

6 Conclusion

This paper described a more realistic and potentially environmentally friendly ground movement decision support system, compared to previous approaches. The framework is designed to combine the runway sequencing problem and ground movement problem, aiming for better global solutions, although only the ground movement element was considered in this paper. This work extends the basic ground movement problem of minimising the travel times to include the concept of absorbing possible waiting times for departures at the gate/stand, to reduce the fuel burn and environmental impact. The sequential QPPTW algorithm which was described here is based on graph theoretical concepts and can include restrictions such as limitations of which taxiways aircraft can use, which taxiways block which and when, and turn limitations at taxiway junctions. In addition, the algorithm provides the opportunity to add buffer times for blocking the reserved taxiways for longer than expected, to absorb small delays and schedule disturbances.

Data was used from an entire day of operations at Zurich Airport, the largest hub airport in Switzerland. This data was used to generate more accurate taxi time estimations for each aircraft, using a taxi speed prediction function which was generated from an extensive statistical analysis of the same dataset. These taxi time estimations were then utilised within the QPPTW algorithm to route and schedule the ground movement. The results are very promising and show potential maximum savings in total taxi time of about 23.6% from using the decision support system described here, together with the statistical taxi time prediction system, compared to the actual performance at the airport. Obviously, further research is necessary to determine the amount of buffer time and runway delay which should be utilised to account for any remaining taxi time uncertainty to avoid starving the runways.

The experimental results of the developed decision support approach show average solution times of only a few milliseconds per aircraft, and are, therefore, adequate for the implementation of such a system for real time use at airports.

We intend to investigate various extensions of this work in future, in addition to the combination of the ground movement problem with the runway sequencing problem. Firstly, the QPPTW algorithm enables the possible waiting times to be spread in different ways. In this paper, they were allocated so as to maximise the stand hold time and to better adapt to schedule disturbances, but an alternative approach would be to develop smoother speed profiles for aircraft, using the engine in a more efficient and environmentally friendly way. Secondly, there are potential benefits to be gained from improving the sequence of consideration of aircraft for the sequential routing algorithm. Finally, we would like to perform a similar analysis for different airport layouts, to better understand the effects of the layout upon the best solution approach, but it will be necessary to obtain more data and support from other airports in order to do so.

Acknowledgements The authors wish to thank the Engineering and Physical Sciences Research Council (EPSRC) for providing the funding which made this research possible. We would also like to thank Flughafen Zürich AG who provided the real dataset.

References

1. ACI EUROPE: An outlook for Europe's airports - Facing the challenges of the 21st century. Tech. rep., Airports Council International Europe (2010). http://www.aci-europe.org/policyoutlook.pdf
2. Atkin, J.A.D., Burke, E.K., Greenwood, J.S.: TSAT allocation at London Heathrow: The relationship between slot compliance, throughput and equity. Public Transport **2**(3), 173–198 (2010). DOI 10.1007/s12469-010-0029-2
3. Atkin, J.A.D., Burke, E.K., Greenwood, J.S.: A comparison of two methods for reducing take-off delay at London Heathrow airport. Journal of Scheduling (2011). DOI 10.1007/s10951-011-0228-y
4. Atkin, J.A.D., Burke, E.K., Greenwood, J.S., Reeson, D.: Hybrid metaheuristics to aid runway scheduling at London Heathrow Airport. Transportation Science **41**(1), 90–106 (2007). DOI 10.1287/trsc.1060.0163
5. Atkin, J.A.D., Burke, E.K., Maathuis, M.H., Ravizza, S.: A combined statistical approach and ground movement model for improving taxi time estimations at airports. Submitted.
6. Atkin, J.A.D., Burke, E.K., Ravizza, S.: The airport ground movement problem: Past and current research and future directions. In: Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT), Budapest, Hungary (2010)
7. Balakrishnan, H., Jung, Y.: A framework for coordinated surface operations planning at Dallas-Fort Worth International Airport. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference, Hilton Head, SC, USA (2007)
8. Burgain, P., Feron, E., Clarke, J.P.: Collaborative virtual queue: Benefit analysis of a collaborative decision making concept applied to congested airport departure operations. Air Traffic Control Quarterly **17**(2), 195–222 (2009)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (2001)
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
11. Dorndorf, U., Drexl, A., Nikulin, Y., Pesch, E.: Flight gate scheduling: State-of-the-art and recent developments. Omega **35**(3), 326–334 (2007)
12. Gawrilow, E., Köhler, E., Möhring, R., Stenzel, B.: Dynamic routing of automated guided vehicles in real-time. In: Mathematics - key technology for the future, pp. 165–178. Springer (2008)
13. Idris, H.R., Clarke, J.P., Bhuva, R., Kang, L.: Queuing model for taxi-out time estimation. Air Traffic Control Quarterly **10**(1), 1–22 (2002)
14. Lesire, C.: Iterative planning of airport ground movements. In: Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT), Budapest, Hungary (2010)
15. Marín, Á.: Airport management: Taxi planning. Annals of Operations Research **143**(1), 191–202 (2006)
16. Marín, Á., Codina, E.: Network design: Taxi planning. Annals of Operations Research **157**(1), 135–151 (2008). DOI 10.1007/s10479-007-0194-0
17. Rappaport, D.B., Yu, P., Griffin, K., Daviau, C.: Quantitative analysis of uncertainty in airport surface operations. In: Proceedings of the AIAA Aviation Technology, Integration, and Operations Conference (2009)
18. Ravizza, S.: Control of automated guided vehicles (AGVs). Master's thesis, ETH Zurich, Switzerland (2009)
19. Roling, P.C., Visser, H.G.: Optimal airport surface traffic planning using mixed-integer linear programming. International Journal of Aerospace Engineering **2008**(1), 1–11 (2008). DOI doi:10.1155/2008/732828
20. Sedgewick, R., Vitter, J.S.: Shortest paths in euclidean graphs. Algorithmica **1**(1), 31–48 (1986). DOI 10.1007/BF01840435
21. Stenzel, B.: Online disjoint vehicle routing with application to AGV routing. Phd thesis, TU Berlin, Germany (2008)