

Solving the flight perturbation problem with meta heuristics

Tobias Andersson

Submitted in December 2004 and accepted by Anthony Cox in September 2005 after 1 revision
© Springer Science + Business Media, Inc. 2006

Abstract When there is a perturbation in a carefully constructed aircraft schedule, e.g. an aircraft breakdown, it is important to minimize the negative consequences of this disturbance. Here, a tabu search and a simulated annealing approach to the flight perturbation problem are presented. The heuristics use a tree-search algorithm to find new schedules for the aircraft, and utilize a path relinking strategy to explore paths between structurally different solutions. The computational results indicate that the solution strategies, especially the tabu search, can be successfully used to solve the flight perturbation problem.

Keywords Aircraft recovery · Irregular operations · Operational airline scheduling · Simulated annealing · Tabu search · Path relinking

Aircraft breakdowns or absent crew members are only two examples of occurrences that might force an airline to make alterations in the aircraft schedule. The flight controllers at the airline operations control center (AOCC) that handle these disruptions, and thus solve the *flight perturbation problem* (FPP), have a number of different actions at their disposal. They may *cancel* flights, in which case the passengers have to be assigned to a new flight and probably compensated in some way. More commonly used actions are to *delay* flights and *swap* flights, i.e. let another aircraft than originally planned operate the flight. It may also be possible to use *spare aircraft* if there are any available, or to *ferry* aircraft, i.e. fly a plane empty to an airport where it is needed.

Cancellations and delays directly affect the passengers and therefore induce costs for reassignments, compensations and badwill along with the operational costs that occur as a result of any schedule changes. The problem of taking care of passengers that are affected by the perturbation and the consequential schedule changes is sometimes called the *passenger recovery problem*. Any alterations in the aircraft schedule may affect the maintenance planning for the aircraft. If an aircraft is assigned additional flights, it is possible that the

T. Andersson

Department of Science and Technology, Linköpings Universitet, SE-601 74 Norrköping, Sweden
e-mail: toban@itn.liu.se

need for a maintenance check will arise sooner than planned. It is therefore necessary for a maintenance planner to approve of any changes to the aircraft schedule before they are executed. The crew schedule may have been disrupted by the same perturbation that caused trouble in the aircraft schedule. Furthermore, any changes in the aircraft schedule may also affect the crew schedule. The *crew recovery problem* is the problem of repairing the crew schedule, e.g. by repositioning crew and utilizing standby crew.

It is possible to attack these perturbation problems all at once, but a more common approach is to solve each one of them separately. In the process, solution information from one of the problems can be used as input data to the other problems. For example, the FPP, which is also called the aircraft recovery problem, might be solved first. This solution can be presented to the people responsible for maintenance planning, crew planning and passenger recovery, who may modify it if possible to suit their requirements, after which the new schedule is executed. If it is not possible to modify the solution, a new schedule has to be created and the process has to be repeated. Most research devoted to airline perturbation problems make the assumption that the planning processes are separated, but the trend is moving towards the study of integrated solution procedures.

The ideal approach would be to solve the FPP while guaranteeing that the maintenance and crew planning stays feasible. Furthermore, the solution should minimize the both aircraft and crew recovery costs as well as the costs for passenger recovery. However, this problem is extremely complex and would require a long solution time. One important quality of a decision support tool that solves the FPP is that it has to produce a solution fast. The disruptions in the schedule will become more serious with time if no changes are made to prevent this from happening. Thus, the controllers have to make very quick decisions, in most situations, and cannot wait a long time for a solution. It is difficult to state a minimum response time. Løve et al. (2002) claim that it should be less than three minutes, but this naturally varies depending on the airline, the severity and urgency of the problem, and on the quality of the solution produced by the decision support tool. If the tool only solves one of the schedule repair problems, it may be necessary to run it multiple times before an acceptable solution is obtained. In this case, it is vital that the solution time is short.

Another important quality is that the solutions should be easy to understand and intuitively plausible from a user perspective, otherwise the controller may dismiss the solution simply because it does not appear valid. A solution should therefore have as few changes as possible from the original schedule. It should also guarantee that the schedule returns to normal after a certain time.

The FPP has received some attention from researchers. Previous attempts to solve it include the works of Teodorovic and Gubernic (1984), Teodorovic and Stojkovic (1990, 1995), Jarrah et al. (1993), Cao and Kanafi (1997a, 1997b), Yan and Yang (1996), Yan and Tu (1997), Argüello, Bard, and Yu (1997), Thengvall, Bard, and Yu (2000), Løve et al. (2002), Filar, Manyem, and White (2001), Stojkovic et al. (2002), Rosenberger et al. (2002), Rosenberger, Johnson, and Nemhauser (2003), and Kohl et al. (2004) among others.

In this paper, a decision support tool for the FPP is presented. A solution is a revised aircraft schedule, where flights may have been delayed, cancelled or swapped. Crew, maintenance and passenger recovery are not regarded explicitly. The solutions always guarantee a return to the original aircraft schedule however, which to some extent ensures that the negative impact on the other schedules is reduced. In Andersson and Värbrand (2004), different solution strategies for the problem are evaluated, including a Lagrangian heuristic and two heuristic column generation algorithms. Here, two meta heuristics that utilize path relinking, are presented and compared to one of the solution strategies proposed in Andersson and

Värbrand (2004). Furthermore, the heuristics in this paper produce not just one solution, but a set of ranked, structurally different solutions that a flight controller can choose from.

1. Modeling assumptions

The problem that is modeled here, is finding a revised aircraft schedule when the original schedule has been perturbed. In order to do this, flights can be cancelled or delayed. New aircraft assignments, i.e. swaps, can be made, both within a fleet of aircraft as well as between different aircraft types. However, a flight can only be transferred from one aircraft type to another if the new aircraft has sufficient capacity to handle the passengers on the flight.

The flight perturbation problem originates from a non-planned event, e.g. an aircraft malfunction. As soon as news of this event reaches the AOCC, the problem has to be solved. Here, the point in time when the problem has to be solved is called the *start time*. From the start time and for a specific period forward, changes can be made in the schedule. We call this period the *decision period*. The decision period ends at the *end time*. At this time it is made sure that the schedule has returned to normal, i.e. each aircraft operates its originally planned flights. By forcing the schedule back to normal, the consequences for the crew planning and the maintenance planning should not be as severe as they would be if the changes made during the decision time had a lasting impact on the schedule. Furthermore, getting back to the plan is common airline practice (Kohl et al., 2004). Therefore, the solutions should be attractive from a user perspective.

The decision time has to be long enough for it to exist good solutions to the problem. However, increasing the decision time also increases the number of flights that has to be regarded in the solution process, and consequently the solution time. It is therefore important to choose the length of the decision time carefully.

At the start time, each aircraft is assigned to a specific route, i.e. a certain sequence of flights that the aircraft is scheduled to operate during the decision time. Each flight that is included generates revenue to the route. The only costs that are regarded here are the costs associated with possible changes or perturbations in the schedule. This means that a route that is not perturbed does not have any associated costs. A route that has been changed in some way, e.g. includes delayed flights, will have a perturbation cost that is subtracted from the revenue.

There are principally two ways of calculating the revenue for a route; by using *real* costs or by using *weights*. When using real costs, the revenue for operating flight j with aircraft k should include the net revenue for operating the flight subtracted by the cost for using a new aircraft if k was not supposed to fly j originally (i.e. a swap cost). The cost for delaying or canceling flight j , should include a badwill cost as well as costs for reassigning and compensating passengers. Many of these costs are of course very hard to obtain, e.g. a badwill cost can never be more than an estimation. Furthermore, neither the net revenue for a flight nor the costs associated with the possible cancellation or delaying of the flight are known in advance. The expected number of passengers on the flight may not be equal to the number of passenger actually flying, due to e.g. refundable tickets, no-shows and overbooking. The main advantage with using real costs, if such can be found, is that it is easy to motivate the optimal solution for the users of the model since the solution is the most cost effective way of dealing with the problem at hand.

When weights are used, the controller will set the costs for the model, e.g. the user may weight the cost for aircraft swaps within the same fleet to 1, for aircraft swaps between fleets to 20, for canceling a flight to 10 and for delaying a flight to 1. Using these weights, the

easiest way to obtain the revenue generated if aircraft k flies flight j , is to start by letting the revenue for operating flight j be equal the cancellation weight multiplied by the number of passengers on that flight. Thus, the revenue is not a measure of how much the passengers pay for their tickets, but more of an inverted shortage cost for not operating a certain flight. In practice, the revenue also depends on whether or not the passengers are able to make their connections. To capture this fact, a cost for a missed passenger connection can be added. Unfortunately in this work, the available data does not include passenger connections, and therefore these potential costs are not regarded.

From the revenue, the swap weight is subtracted if aircraft k was not originally assigned to that flight. A swap is thus defined here as a flight that is operated by another aircraft than the originally planned. It should be noted that the swap costs are the components that effectively preserve large parts of the original schedule. From the revenue, the cost for delaying flight j also has to be subtracted. This cost is calculated as the weight for delays multiplied by the number of passengers on flight j and by the number of time units that the flight is delayed.

In a feasible solution, each aircraft is assigned to a feasible route. The value of the solution is calculated as the sum of the revenues for the routes. If the weight for swaps is set to a large value while the weight for delay is set to a low value, a solution with delayed passengers will probably rank higher than a solution with many swaps. Still, even if the flight controller has set the weights, it may be interesting for him or her to obtain a set of structurally different solutions to choose from. None of these solutions should however be dominated by another. Let c_i be the number of passengers affected by a cancellation in solution i , s_i the number of regular swaps, f_i the number of swaps between aircraft types and d_i the delay in passenger minutes (number of passengers on a flight multiplied by the delay for the flight) for solution i . Then solution i dominates solution j if:

$$c_j \geq c_i \quad \text{AND} \quad s_j \geq s_i \quad \text{AND} \quad f_j \geq f_i \quad \text{AND} \quad d_j \geq d_i \quad \text{AND} \\ (c_j > c_i \quad \text{OR} \quad s_j > s_i \quad \text{OR} \quad f_j > f_i \quad \text{OR} \quad d_j > d_i)$$

Of course, solutions that are not dominated may still be much less attractive from a user perspective. The weights can be used to rank all, or a certain number of the best, non-dominated solutions that are found by a solution algorithm. These solutions can then be presented to the user who will have a set of distinctly different solutions to choose from. In this way, the experience of the user can be efficiently utilized, as she or he is able to judge the solutions produced by the model and add relevant aspects that are not covered by it, e.g. the possibility to reassign passengers from a cancelled flight to one that is not cancelled.

2. Two meta heuristics

A meta heuristic might be described as an enhanced local search algorithm, where the main improvement lies in the ability to move away from local optima. To use a local search, it is necessary to have a solution to start with. By performing a *change* in this solution, a new solution is obtained. The solutions that can be reached from the initial solution by a certain change are called *neighbors*, and the total set of neighbors is referred to as the *neighborhood*. In a local search, the neighborhood (or a part of it) is explored, and the best neighbor is selected as the improving solution. If no better solution can be found in the neighborhood, the algorithm terminates.

Algorithm 1. *The local search algorithm*

1. Start with a feasible solution, s , with an objective function value of z
2. Let F_{sk} be the set of flights assigned to aircraft k in solution s
3. Let N be the set of flights not assigned to any aircraft
4. Let $LocalOpt = 0$
5. **Repeat** {
6. Let $z^{temp} = z$, $N^{temp} = N$, $s^{temp} = s$
7. Let $P =$ the set of unique aircraft pairs in s
8. **While** $P \neq \emptyset$ {
9. Select an aircraft pair in P : aircraft a and b
10. $P = P - (a, b)$
11. $N^{temp} = N^{temp} \cup F_{sa} \cup F_{sb}$
12. Find all feasible routes for aircraft a , using only the flights in the set N^{temp}
13. Repeat the step above for aircraft b
14. Pick one route for each of the two aircraft so that no flight is covered more than once, and the revenue is maximized
15. Let s_{new} be the solution with the two new routes, z_{new} the value of this solution and N_{new} the set of cancelled flights in this solution
16. **If** $z_{new} > z^{temp}$ { $s^{temp} = s_{new}$, $z^{temp} = z_{new}$, $N^{temp} = N_{new}$ }
17. }
18. **If** $z^{temp} \neq z$ { $s = s^{temp}$, $z = z^{temp}$, $N = N^{temp}$ }
19. **Else** { $LocalOpt = 1$ }
20. } **Until** $LocalOpt = 1$

Algorithm 1 describes a local search algorithm for finding solutions to the FPP. This local search provides the base for the two meta heuristics. In Step 1 a feasible solution is needed. Here, the solution where only delays are used to resolve the perturbation is selected as the initial solution. The set F_{sk} in Step 2 is the route for aircraft k in the initial solution. The set N in Step 3 will be empty in the beginning, when using the initial solution with only delayed flights, but will contain the possible cancelled flights in the current solution when the algorithm has run for a couple of iterations.

The main loop of the local search algorithm presented in Algorithm 1 starts at Step 5 and ends at Step 20 when no improving solution can be found. In each iteration, the neighbors to the current solution, s , are evaluated; this is done in the loop starting at Step 8. Two aircraft are selected in some way, and all the flights they are supposed to fly in the current solution are added to a pool together with the cancelled flights (Step 9–11). In Step 12, a recursive tree-search algorithm is used to find all interesting, feasible routes for aircraft a , assuming that the aircraft can only operate flights in the pool, N^{temp} .

The routes that are found by the tree-search algorithm have to be feasible from a connection perspective, e.g. if an aircraft operates flight 2 after flight 1, the departure airport for flight 2 has to be the same as the arrival airport for flight 1. Furthermore, the last flight in the route has to be connectable to the originally planned schedule for the aircraft. Sometimes it is necessary to delay a flight before inserting it in a route. If this is the case, the minimum required delay is used. Using excessive delay when connecting the flight creates a route that is dominated by one where minimum delay is used. Thus, such a route is not interesting, even though it is still feasible. The routes also have to be feasible from a capacity perspective. If the number of passengers on a flight exceeds the capacity of the aircraft, which may happen when trying to perform swaps between different aircraft types, the swap is not allowed.

The number of routes produced by the tree-search algorithm varies with the problem size and the data characteristics. In the computational tests performed in this work, the number of columns that was produced for an aircraft varied from two up to thousands, but the median value was around 20.

The route finding process is repeated for aircraft b , and in Step 14, one route is selected for each one of the two aircraft to form the new solution. In Step 14, a small set packing problem is actually solved; a column has to be assigned to each of the aircraft and no flight must be flown more than once (but might be cancelled). However, no optimization techniques are used to solve it, since complete enumeration of the possible combinations is quite efficient in this case. In Step 16, the new solution is checked against the current best neighbor candidate, and if it is better (i.e. has a higher revenue) it is saved. If the objective function value for the best neighbor to s is higher than the current objective function value (z), the local search will move to this new solution, otherwise the flag *LocalOpt* is set to 1 and the search terminates (Step 18–20).

The neighborhood in the local search is defined by all solutions that can be reached by picking a unique pair of aircraft and finding new routes for them. This means that if K is the number of aircraft, there are $K(K - 1)/2$ solutions in the neighborhood. A local search could check all the solutions in the neighborhood and pick the best one, before making the move to the new solution, like in Algorithm 1. If this is too computationally expensive, it is possible to select the aircraft pair e.g. randomly and always move when a better solution is found. Algorithm 1 can easily be extended to the meta heuristics tabu search (TS) and simulated annealing (SA).

2.1. Tabu search

The classic reference for tabu search is Glover (1986), but today information on the heuristic can be found in many textbooks, e.g. Aarts and Lenstra (1997) and Reeves (1993). The enhancement built into a TS algorithm, i.e. what separates it from a local search algorithm, is the addition of a *tabu list*. The tabu list consists of certain attributes that define if a move from a certain solution to another is allowed. The main purposes are to assure that the algorithm does not stop at a local optimum, and to prevent cycling, i.e. that the same solutions are visited repeatedly. If the tabu list consists of the L latest solutions, it is possible to guarantee that cycles shorter than L iterations will not occur. However, in many cases it is impractical to use the actual solutions as the tabu attribute. This requires each candidate solution to be constructed and evaluated before it is possible to determine the tabu status of the solution, a process that may be quite time consuming. Therefore the tabu list often consists of the latest L moves, i.e. the latest changes that have been performed. Restricting a certain change in the solution, usually affects more than one solution in the neighborhood. This means that forbidding a certain change is more restrictive than forbidding a certain solution. However, if no information of the recently visited solutions is saved (with the obvious exception of the best solution found) it is not longer possible to guarantee that cycles will not appear. Therefore, it is important to choose the tabu attribute carefully.

Many would consider the most obvious choice of a tabu attribute for the problem studied here, to be an aircraft or an aircraft pair. In the first case, this means that an aircraft that has already been selected must not be chosen again for the next L iterations. In the second case, it is forbidden to select a certain aircraft pair. The first choice means that a large portion of the solutions in the neighborhood will become tabu, while the second does not even guarantee the avoidance of cycles longer than three iterations (see Table 1).

In Table 1, aircraft A and B obtain new routes in iteration 1; aircraft A changes from route $a1$ to route $a2$ and aircraft B from route $b1$ to route $b2$. Then aircraft A and C are selected in

Table 1 Example of cycling when using inappropriate tabu attributes

	A		B		C	
	old	new	old	new	old	new
0	–	a1	–	b1	–	c1
1	a1	a2	b1	b2	c1	c1
2	a2	a1	b1	b2	c1	c2
3	a1	a1	b2	b1	c2	c1

iteration 2, where the best route for A is the old one, *a1*, and C gets a new route, *c2*. Finally in iteration 3, both aircraft B and C obtain their old routes, *b1* and *c1*, and the start solution, the same as in iteration 0, is reached.

To pick an aircraft or an aircraft pair as tabu attributes are examples of making a change in the solution tabu, rather than making a solution tabu. When implemented, initial testing shows that these types of tabu attributes are not suited for the problem studied here. Therefore, an attempt is made to make solutions tabu instead of changes, but this requires that it is possible to efficiently check whether or not a solution is tabu. A solution is defined by which routes the aircraft fly. Thus by checking the assignments for the aircraft, it is possible to see if the solution has been visited recently. However, this would not be a very quick way of finding out if a solution is tabu. Instead, let the value of the solution, the accumulated number of swaps, aircraft type swaps and the total accumulated delay experienced by the passengers define a unique solution. This makes it less time consuming to compare two solutions. What may happen in rare cases is that all values match for two different solutions, but in this case they have very similar characteristics, and the performance of the algorithm most likely benefits from making similar solutions tabu. This means that when a new solution is made tabu according to the parameters described, not only the solution just visited is made tabu, but all solutions sharing its characteristics as well.

The main change in Algorithm 1 is that a new solution, s_{new} , is checked for tabu status before a move to this solution is considered, e.g. between Step 14 and 15. It will also be possible to accept solutions that are worse than the current one, which means that the algorithm will always move to the new solution, i.e. the *if* check in Step 18 is removed. Finally z^{temp} will have to be initialized to a large negative value in Step 6, and an alternative termination criterion to the one in Step 20 is needed, e.g. a fixed number of iterations or a number of iterations without improvement.

2.2. Simulated annealing

Simulated annealing is constructed to imitate the cooling of material from a fluid to a solid state. Kirkpatrick, Gellat, and Vecchi (1983) suggested that by simulating this process, solutions to optimization problems could be found. Since then SA has been extensively studied theoretically and widely used in practical applications.

Two important qualities of all SA algorithms are the temperature and the cooling schedule. The temperature, t , controls how likely it is for the search to move to a solution worse than the current; a high t corresponds to a high probability. If a high initial t is used, the search will move through the search space in a randomly fashion, but this will hardly guarantee a convergence towards a good solution. Therefore, it is necessary to lower the temperature during the iterative process and in this way lower the probability for accepting worse solutions. This is done with a cooling schedule. Common for most cooling schedules is that they make

the temperature converge towards zero, which means that the algorithm will become a local search as no solutions worse than the current will be accepted. The two most widely used cooling schedules are (Dowsland, 1993):

- Reduce t according to $t = \alpha t$, every $nrep$ iteration, e.g. using an α between 0.8 and 0.99
- Reduce t according to $t = t/(1 + \beta t)$, every iteration, using a small β , e.g. 0.01

A worse solution is accepted with a probability that depends on the temperature and on how much lower the revenue is. The probability is calculated as $\exp((z_{\text{new}} - z)/t)$ and checked against a randomly generated number between 0 and 1. If the probability is higher than the randomly generated number, the move is made, despite the new solution being worse than the current.

In SA, a solution in the neighborhood is selected randomly and evaluated. As explained above, the algorithm may move to this solution or not depending on the value of the solution and the temperature. Some changes have to be made to Algorithm 1 in order to implement this. The *while* loop in Step 8 can be removed since only one solution will be evaluated each iteration. This also eliminates the need for the *temp* variables, i.e. a neighbor is selected randomly (Step 9) and evaluated (Step 15). If it is accepted, the search moves to this solution, otherwise the search stays at the current solution.

When the temperature has been reduced significantly, no solution that is worse than the current will be accepted. If the current solution is a local optimum, this means that the search will not continue. To prevent this, a counter is added, and when no new solution has been accepted for a certain number of iterations (here three times the size of the neighborhood), a local search mode is triggered. In the local search mode, all neighbors to the current solution are examined. If a move to one of them is accepted, the SA search continues as usual. If the search does not move to any of the neighbors, the current solution is verified as a local optimum, and temperature is raised to the start value. This strategy does not work however, if there are multiple local optima. Then the local search will always find the equally good solutions and move to one of them. To identify this behavior, another counter keeps track of the objective function value of the accepted solutions. When only solutions with the same objective function value have been accepted for a certain number of iterations (here the neighborhood size divided by three), the temperature is raised to the start value. The same type of termination criterion as for the TS heuristic can be used for SA.

2.3. Path relinking and non-dominated solutions

Path relinking (PR) was introduced by Glover and Laguna (1993) as an extension in tabu search. Two solutions, one initial solution and one guiding solution, are selected from a set of elite solutions that have been found during the search. From the initial solution to the guiding solution, a path is generated. The idea is that the solutions in the path should share some characteristics with the initial and guiding solutions. In the best case, a new solution improving the current best may be found among the intermediate solutions, but even if not, one of the intermediate solutions can be selected as a new starting point for the search.

Three components are vital in PR (Ho and Gendreau, 2004): how to construct the set of elite solutions, how to choose the initial and guiding solutions, and how to move along the path.

The set of elite solutions can e.g. consist of local optima, or solutions that differs significantly. In the application studied here, a set of non-dominated solutions, which by definition differs significantly, is desired as an output from the search. This set is therefore appropriate to use as the set of elite solutions.

There exist a number of strategies for choosing initial and guiding solutions, e.g. pick the best and the worst solutions in the set, or select the solutions randomly. Here, initial testing shows that the set of non-dominated solutions rarely grow beyond an unmanageable size. In general, it stays below 10–15 solutions. However, to prevent the set from growing too large, no more than 20 non-dominated solutions are stored in the set. Of course, the non-dominated solutions with the highest objective function values are used. It is not particularly time consuming to perform a path relinking between two solutions. Therefore, as soon as a new solution enters the elite set, it is selected as the initial solution. Path relinking is then performed to all the other solutions in the set.

Depending on how the moves from the initial to the guiding solution are performed, there exist a variety of possible paths. The objective in each move is to bring some attributes from the guiding solution to the current solution. How the moves are accomplished in the application studied here is described in Algorithm 2.

Algorithm 2. *Path relinking*

1. Assume that there exist an initial solution s and a guiding solution t
 2. Let the current solution, $x = s$
 3. Let A = the set of aircraft, $k \in A$
 4. Let F_{xk} = the set of flights assigned to aircraft k in solution x
 5. Let r_{xk} = the revenue produced by aircraft k in solution x
 6. Let N_x be the set of flights not assigned to any aircraft in solution x
 7. **While** ($x \neq t$) {
 8. Let $y = x$
 9. Let aircraft $a = \max_{k \in A} \{ |r_{xk} - r_{tk}| \}$
 10. Let aircraft $b =$ an aircraft where $F_{ta} \cap F_{xb} \neq \emptyset$, if such an aircraft exist, else $b = \emptyset$
 11. **If** ($b = \emptyset$) {
 12. $F_{ya} = F_{ta}$
 13. $N_a = (F_{xa} \cup F_{ya}) - (F_{xa} \cap F_{ya})$
 14. $N_y = (N_a \cup N_x) - (N_a \cap N_x)$
 15. **}Else** {
 16. Find all feasible routes for aircraft a , using only flights in the set $Q_a = F_{ta} \cap (F_{xa} \cup F_{xb} \cup N_x)$
 17. Let R_a be the set of routes produced in the step above, let z_i be the revenue for route i , and let F_i be the set of flights in route i , $i \in R_a$
 18. $F_{ya} = \{F_i | i = \min_{j \in R_a} \{ |z_j - r_{ta}| \} \}$
 19. Find all feasible routes for aircraft b , using flights in the set $Q_b = (F_{xa} \cup F_{xb} \cup N_x) - F_{ya}$
 20. Let R_b be the set of routes produced in the step above, let z_i be the revenue for route i , and let F_i be the set of flights in route i , $i \in R_b$
 21. $F_{yb} = \{F_i | i = \min_{j \in R_b} \{ |z_j - r_{tb}| \} \}$
 22. $N_y = (F_{xa} \cup F_{xb} \cup N_x) - F_{ya} - F_{yb}$
 23. **}**
 24. $x = y$
 25. **}**
-

Algorithm 2 starts with an initial solution and a guiding solution. In each iteration of the loop, beginning in Step 7, a new intermediate solution, y , is found and stored. If the new solution is non-dominated, it is also saved as one of the elite solutions. If the new solution dominates any of the old elite solutions, the old solution is removed from the set. The same procedure is used for updating the set of elite solutions during the normal search (TS or SA) as well.

In Step 9, an aircraft is selected. From the set of all aircraft, the one whose route revenue differs mostly from the route revenue in the guiding solution is chosen. The reason for this is to find a move that results in a long step towards the guiding solution. The move consists of finding a new route for aircraft a . This new route should have more flights in common with the route in the guiding solution. In Step 10, one additional aircraft, b , is selected, if there exist any aircraft operating flights that should be operated by a , in accordance to the guiding solution. If no such aircraft exist, aircraft a operates flights that should be cancelled, or there exist cancelled flights that should be operated by aircraft a . In both cases, aircraft a is assigned the route it has in the guiding solution, and any additional flights are added to the set of cancelled flights, N_y (Step 12–14). Similarly, any missing flights are taken from the set of cancelled flight.

If there exist an aircraft b , all the flights from aircraft a and b and the set of cancelled flights are added to a pool. From this pool, only the set of flights that can be found in the route for aircraft a in the guiding solution is used to create new routes (Step 16), using the same tree-search algorithm as in Algorithm 1. The revenues of the routes in the new set, R_a , are compared to the revenue of the route in the guiding solution. The route whose revenue is closest to the revenue in the guiding solution, is selected as the new route for aircraft a (Step 17–18). In a similar manner, a new route is generated for aircraft b (Step 19–21). The set of cancelled flights is then updated to reflect the new routes for aircraft a and b (Step 22).

In Step 24, the current solution, x , is updated with the changes. The process continues until the guiding solution has been reached, i.e. $x = t$.

When the path relinking process has finished, it is possible to use one of the intermediate solutions as the starting point for the continuing TS or SA search. However, initial experiments have shown that this does not improve the search for solutions with a high objective function value. Thus, the only time the regular search is deviated to one of the intermediate solutions is when the new solution is the best found so far.

3. Computational results

For the computational tests, there are three different data sets, $s1$, $s2$, and $s3$, of different sizes. Table 2 sums up the important characteristics for the data sets.

Table 2 The data sets

	Data Set		
	$s1$	$s2$	$s3$
Number of aircraft	13	30	58
Number of aircraft types	2	5	5
Number of flights	98	215	436
Number of airports	19	32	41

The first two data sets are flight data from a Swedish domestic airline. The third data set is based on the same data, but artificially extended in order to get a larger set. The data sets are perturbed in two different ways, a and b , where a is an aircraft breakdown and b the imposed delay on a number of flights in the set, e.g. caused by a ground holding program (some flights are issued a new departure time by the air traffic control authorities in order to reduce congestion). This gives the data sets $s1a$, $s1b$, $s2a$, $s2b$, $s3a$, and $s3b$. In $s1a$ an aircraft is unavailable for five hours, and in $s1b$ two flights are imposed with delays for 25 and 30 minutes respectively. In $s2a$ an aircraft is unavailable for six hours, while in $s2b$ four flights are imposed with delays ranging from 15 up to 40 minutes. In $s3a$, one aircraft is unavailable for ten hours, and two flights are delayed for one hour each. Finally, in $s3b$, all flights departing from the largest airport are affected by a decrease in capacity at the airport. This results in 55 flights being delayed from five minutes up to 4 hours and 35 min.

By manipulating the weights, i.e. the objective function coefficients in the problem, even more problem instances can be obtained from the data sets. $s1a1$ will then correspond to a problem using flight data $s1$, with perturbation a , and weight setting 1, which for $s1a1$ correspond to a cancellation weight (can) of 20, a weight for swaps between different aircraft types (sw-f) of 100, a swap weight (sw) of 20 and a delay weight (del) of 1 (see Table 3). Each weight setting has been selected to produce a unique solution (BKS) with a different characteristic than the other solutions. For example, in run $s1a1$ the solution includes 30 cancelled passengers (c), 2 swaps between different aircraft types (f), 8 regular swaps (s) and no delayed flights (d). Now, if the flight controller does not like swaps between aircraft types, the corresponding weight may be increased from 100 to 1000, which gives the setting in run $s1a2$. In run $s1a2$ the solution includes no swaps between aircraft types, but instead the number of passengers affected by a cancelled flights has increased to 46. The delay (d), which is evident in $s1a3$ where the cancellation weight has been increased as well, is given in passenger minutes.

The best known solutions (BKS) for all runs on the data sets $s1$ and $s2$ have been obtained by a Lagrangian heuristic (Andersson and Värbrand, 2004), which also produce an upper bound (UBD) for the solutions. The gap between the BKS and the UBD is small enough to ensure that the optimal solution has been found for all runs on these sets. Unfortunately, the Lagrangian heuristic requires too much internal computer memory for it to solve any instances on the data set $s3$. The BKS's for this data set have been obtained by running each of the TS and SA algorithms for one hour each and take the best solution found. The values in the z^d column are for solutions that use only delays to resolve a perturbation. These can be used as reference values, as the large objective function values make any kind of gap between the solutions rather small. All test have been performed on a on a PC laptop with a Pentium 1000 MHz processor and 512 Mb RAM.

The maximum delay allowed for a single flight is set to 90 minutes for all flights and a feasible route may contain a maximum of 16 flights. The decision time is one day; the first flights start a couple of minutes after four am, and the last flights arrive to their overnight station around nine pm, if there are no delays.

The start temperature for the SA algorithm should be high enough to enable almost any move in the beginning. Therefore it is set to the mean revenue that is produced by two aircraft. This means that even a solution that cancels all flights operated by the two selected aircraft probably will be accepted. The temperature is then reduced according to a geometric cooling schedule, $t = t * a$, where $a = 0.78$, every $nrep$ iteration. $nrep$ is set equal to the size of the neighborhood i.e. 78, 435, and 1653 for $s1$, $s2$, and, $s3$ respectively—to give the algorithm plenty of time to explore each temperature.

Table 3 Characteristics of the best known solutions for each problem

<i>D-set</i>	Weights				Results						
	can	sw-f	sw	del	UBD	z^d	BKS	c	f	s	d
<i>s1a1</i>	20	100	10	1	43840.8	28685	43840	30	2	8	0
<i>s1a2</i>	20	1000	10	1	43760.3	28685	43760	46	0	4	0
<i>s1a3</i>	100	1000	10	1	218980.0	207565	218980	32	0	0	1420
<i>s1a4</i>	100	100	10	1	220680.9	207565	220680	25	4	2	0
<i>s1b1</i>	20	100	10	1	44600.1	43140	44600	0	0	12	0
<i>s1b2</i>	20	100	100	1	44070.0	43140	44070	0	0	4	250
<i>s1b3</i>	20	100	400	1	43140.0	43140	43140	0	0	0	1580
<i>s2a1</i>	20	100	10	1	68480.9	38020	68480	29	2	4	0
<i>s2a2</i>	20	1000	10	1	68000.9	38020	68000	65	0	0	0
<i>s2a3</i>	100	1000	10	1	341865.0	315220	341865	29	0	12	1615
<i>s2b1</i>	20	100	10	1	69080.9	68655	69080	0	0	12	100
<i>s2b2</i>	20	100	10	10	68720.8	62850	68720	23	0	12	0
<i>s2b3</i>	20	100	50	1	68900.9	68655	68900	0	2	2	100
<i>s3a1</i>	20	100	10	1	–	62340	141970	85	2	13	0
<i>s3a2</i>	100	100	10	1	–	638340	713940	40	6	14	1320
<i>s3a3</i>	100	1000	10	1	–	638340	713090	40	0	27	2640
<i>s3b1</i>	20	100	10	1	–	127010	134240	256	3	13	4030
<i>s3b2</i>	20	1000	10	1	–	127010	133980	291	0	17	4030
<i>s3b3</i>	20	1000	10	10	–	–25900	130520	628	0	32	60

In the TS, all the solutions in the neighborhood are evaluated in during an iteration, and the best one that is not tabu is selected as the new solution. A tabu list with a length set to the size of the neighborhood divided by twenty plus twenty—i.e. 24, 42, and 103 for *s1*, *s2*, and, *s3* respectively—is used.

The results from the computational tests are shown in Table 4. The *D-set* column tells which one of the four data sets, and which weights, that are used. The *Strat* column states which solution strategy that is used, where *SA* = simulated annealing and *TS* = tabu search. The best solution found within 5, 10, 15, 30 and 60 seconds are compared to the best known solution (*BKS*) for the particular problem.

The results in Table 4 are promising, especially for the TS heuristic. For this solution strategy, the BKS is quickly found for all of the runs on the data sets *s1* and *s2*, except in *s1b1* and *s2b2*, where in the latter case the BKS is found after 20 sec. The largest data set, *s3*, offers more of a challenge, even if the BKS is found in some of these tests as well. The SA heuristic does produce acceptable solutions, but in general seems less successful than the TS heuristic. This is especially evident for the set *s3b*, where the SA algorithm produces solutions with a significantly lower objective function value compared to the TS heuristic. The only problem for which the SA heuristic performs better than the TS heuristic is *s1b1* and the first 15 seconds for *s3b3*.

Table 4 Results from testing the meta heuristics

D-set	Strat	Results [% from BKS]				
		5 s	10 s	15 s	30s	60 s
<i>s1a1</i>	SA	0.025	0.004	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s1a2</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s1a3</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s1a4</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s1b1</i>	SA	0.009	0.002	0. 002	0.000	0.000
	TS	0.045	0.045	0. 045	0.045	0.045
<i>s1b2</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s1b3</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s2a1</i>	SA	0.083	0.072	0. 015	0.004	0.000
	TS	0.058	0.000	0. 000	0.000	0.000
<i>s2a2</i>	SA	0.000	0.000	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s2a3</i>	SA	0.151	0.110	0. 024	0.009	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s2b1</i>	SA	0.197	0.161	0. 056	0.020	0.002
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s2b2</i>	SA	0.177	0.177	0. 148	0.074	0.014
	TS	0.146	0.146	0. 087	0.000	0.000
<i>s2b3</i>	SA	0.039	0.019	0. 000	0.000	0.000
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s3a1</i>	SA	0.561	0.561	0. 561	0.412	0.347
	TS	0.155	0.155	0. 155	0.155	0.155
<i>s3a2</i>	SA	0.477	0.366	0. 234	0.182	0.146
	TS	0.144	0.144	0. 144	0.032	0.000
<i>s3a3</i>	SA	0.459	0.454	0. 182	0.102	0.052
	TS	0.273	0.273	0. 273	0.273	0.186
<i>s3b1</i>	SA	2.858	2.858	2. 858	2.133	1.964
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s3b2</i>	SA	2.756	2.756	2. 752	2.285	2.066
	TS	0.000	0.000	0. 000	0.000	0.000
<i>s3b3</i>	SA	1.570	1.528	1. 503	1.351	0.280
	TS	6.827	1.984	0.061	0.038	0.038

Table 5 Number of non-dominated solutions found after 20 seconds, by tabu search (TS) and tabu search with path relinking (TS-PR)

	s1a				s1b			s2a			s2b			s3a			s3b		
	1	2	3	4	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
TS	5	2	3	3	2	3	4	3	5	4	3	5	7	4	6	4	8	7	14
TS-PR	7	2	4	3	11	4	4	4	5	13	10	17	10	5	20	4	20	20	20

One interesting question is whether the algorithms benefit from the path relinking process. From the computational tests, only one conclusion can be drawn. The path relinking process is valuable for finding non-dominated solutions.

In Table 5, the number of non-dominated solutions found after 20 seconds by the tabu search heuristic is presented, both without and when using path relinking. New non-dominated solutions are often found when moving between two non-dominated solutions. Thus, the number of non-dominated solutions available to the user after 20 seconds is larger when path relinking is utilized. The results did not indicate that the search for solutions with a high objective function value benefited from PR, although no adverse effect could be identified either.

When the SA or TS heuristics have finished, they present a ranked list of non-dominated solutions for the user. This list can look like in Table 6, which is from a 20 sec. run on problem *s2b2*. Solutions 8–16 are removed to save space.

The best solution, as ranked by the weights, is presented as the first solution in Table 6. Interesting to note is that both the best solutions to *s2b1* and *s2b3* can be found in the list, as solutions number 4 and 6. Furthermore, the two extreme solutions where only cancellations are used to resolve the perturbation (Solution 7) and only delays are used (Solution 17), can be found in the list as well. If the user does not agree that the first solution is the most suitable for solving the problem, she or he can consider any other solution in the list, or set new weights and solve the problem again.

As the SA algorithm includes random elements, it may not produce the same solution every time it is run. Therefore the algorithm is run 50 times before a mean objective function value is calculated. For a given solution time –5, 10, 15 seconds etc.—values from the 50 tests are used to calculate mean values (*z*) for the objective function. From this set of values,

Table 6 The list of non-dominated solutions presented by the tabu search heuristic after 20 seconds on problem *s2b2*

Solution	<i>z</i>	<i>c</i>	<i>f</i>	<i>s</i>	<i>d</i>
1	68720	23	0	12	0
2	68620	23	2	2	0
3	68460	41	0	2	0
4	68180	0	0	12	100
5	68100	50	2	0	0
6	68080	0	2	2	100
7	67940	68	0	0	0
:	:	:	:	:	:
17	62850	0	0	0	645

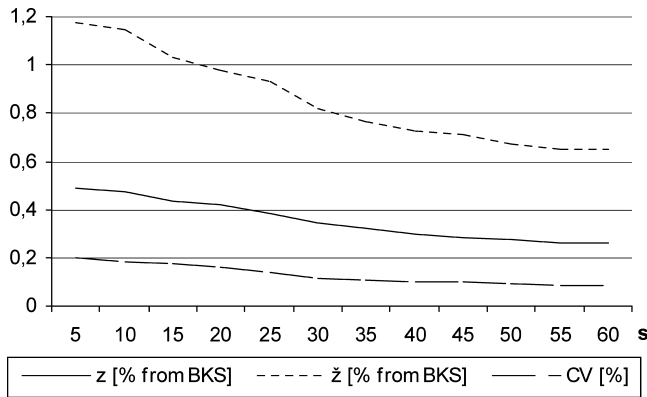


Fig. 1 Results when studying the robustness of the simulated annealing heuristic

the minimum value is also extracted:

$$\bar{z} = \min_{i=1, \dots, 50} \{z_i\}$$

where z_i = the objective function value in run i . Furthermore, the coefficient of variation (CV) is calculated as:

$$CV = 100 \frac{\sigma}{z}$$

where σ is the standard deviation.

In Figure 1, mean values for z , $\bar{z}$ (expressed as the gap to the BKS) and CV over the results from the 19 problems, *s1a1*–*s3b3*, have been calculated. It is clear that the algorithm sometimes produce inferior solutions. Although it is possible for the user to run the algorithm again and hope for a better solution, this is a flaw from a practical perspective. The robustness of the algorithm does increase though, if the algorithm is allowed to run for a longer time. This is indicated by the decrease in the CV, and not especially surprising, since the chance of finding a good solution, i.e. with an objective function value closer to the BKS, increases as the solution time increases.

In Andersson and Värbrand (2004), two heuristics based on column generation were presented. A Dantzig-Wolfe master problem was solved with LP-relaxation and Branch and Bound in the first strategy. In the second, a Lagrangian heuristic was used to solve the master problem. In both strategies, new columns for the master problem were generated heuristically during the solution procedures. The performance for the two heuristics was similar. It is therefore enough to compare the meta heuristics to one of the column generation heuristics; the one utilizing Lagrangian relaxation (LH).

The three different heuristics are compared in Figure 2, where mean values for the objective function values over 19 runs, *s1a1*–*s3b3*, have been calculated. The results indicate that the TS heuristic is the most efficient. Compared to the SA heuristics, the TS also has the advantage of being deterministic, and thus presents the same solution each time it is run for a certain problem. The comparatively weak result for the LH algorithm is mainly due to the difficulty the algorithm has in producing good solutions for the largest data set, *s3*. The LH algorithm suffers most of the three algorithms from the increase in number of flights and aircraft.

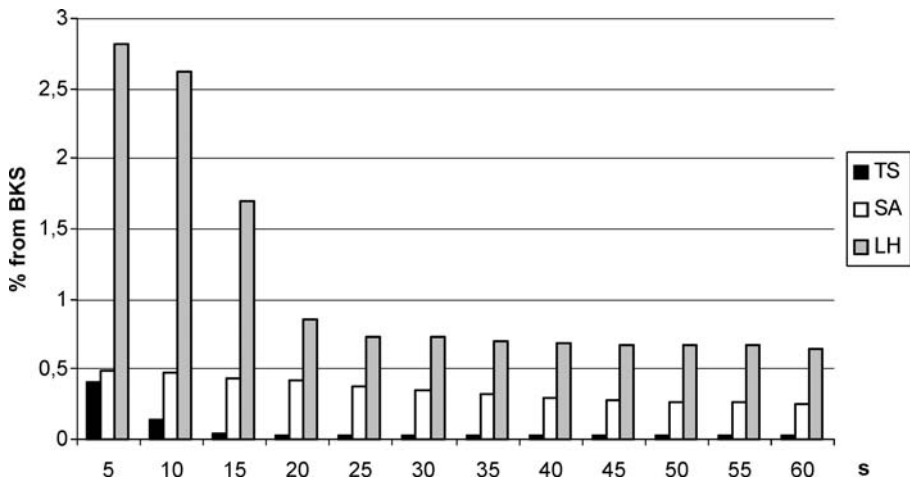


Fig. 2 Comparing the heuristics tabu search (TS), simulated annealing (SA) and Lagrangian column generation (LH)

4. Conclusions

Two meta heuristics, a tabu search and a simulated annealing algorithm both utilizing path relinking, have been developed and tested for the flight perturbation problem. The tabu search heuristic stands out as the preferable solution strategy, both when considering the quality of the solutions and the robustness of the method. Within 15 seconds, the tabu search always produces a solution that is less than 0.3% from the best known solution.

The tool developed in the work is meant for assisting the flight controllers at an airline operations control centers. When a perturbation has disrupted the original aircraft schedule, the tool can be utilized, and will present a ranked set of structurally different solutions to the flight controller. The controller can select one of the solutions in the list, or choose new weights, i.e. new objective function coefficients, and solve the problem again.

Acknowledgements This research was partly sponsored by RM Rocate. The author would like to thank RM Rocate for the opportunity of working with this problem and for supplying real problem data. The author also kindly thanks the referees for valuable comments and suggestions.

References

- Aarts, E. and J. Lenstra, (eds.), (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons Ltd.
- Andersson, T. and P. Värbrand. (2004). "The Flight Perturbation Problem." *Transportation Planning and Technology* 27(2), 91–117.
- Argüello, M., J. Bard, and G. Yu. (1997). "A GRASP for Aircraft Routing in Response to Groundings and Delays." *Journal of Combinatorial Optimization* 1(3), 211–228.
- Cao, J. and A. Kanafani. (1997a). "Real-Time Decision Support for Integration of Airline Flight Cancellations and Delays Part 1: Mathematical Formulation." *Transportation Planning and Technology* 20, 183–199.
- Cao, J. and A. Kanafani. (1997b). "Real-Time Decision Support for Integration of Airline Flight Cancellations and Delays Part 2: Algorithm and Computational Experiments." *Transportation Planning and Technology* 20, 201–217.

- Dowsland, K. (1993). "Simulated Annealing." In Colin R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications pp. 20–69.
- Filar, J., P. Manyem, and K. White. (2001). "How Airlines and Airports Recover From Schedule Perturbations: A survey." *Annals of Operations Research* 108, 315–333.
- Glover, F. (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence." *Computers & Operations Research* 5, 533–549.
- Glover, F. and M. Laguna. (1993). "Tabu Search." In C. Reeves, (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications pp. 70–150.
- Ho, S. and M. Gendreau. (2004). "Path Relinking for The Vehicle Routing Problem." Technical Report, Department of Informatics, University of Bergen.
- Jarrah, A.I., G. Yu, N. Krishnamurthy, and A. Rakshit. (1993). "A Decision Support Framework for Airline Flight Cancellations and Delays." *Transportation Science* 27, 266–280.
- Kirkpatrick, S., C. Gellat, and M. Vecchi. (1983). "Optimization by Simulated Annealing." *Science* 220, 671–680.
- Kohl, N., A. Larsen, J. Larsen, A. Ross, and S. Tiourine. (2004). "Airline Disruption Management—Perspectives, Experiences and Outlook." IMM-Technical Report-2004-16, Informatics and Mathematical Modelling, Technical University of Denmark, DTU.
- Løve, M., K. Sorensen, J. Larsen, and J. Clausen. (2002). "Disruption Management for An Airline—Rescheduling of Aircraft." In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, G. Raidl, (eds.), *Applications of Evolutionary Computing* Springer: Lecture Notes in Computer Science Vol. 2279, pp 315–324.
- Reeves, C. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications.
- Rosenberger, J., A. Schaefer, D. Goldsman, E. Johnson, A. Kleywegt, and G. Nemhauser. (2002). "A Stochastic Model of Airline Operations." *Transportation Science* 36(4), 357–377.
- Rosenberger, J., E. Johnson, and G. Nemhauser. (2003). "Rerouting Aircraft for Airline Recovery." *Transportation Science* 37(4), 408–421.
- Stojkovic, G., F. Soumis, J. Desrosiers, and M. Solomon. (2002). "An Optimization Model for A Real-Time Flight Scheduling Problem." *Transportation Research Part A* 36, 779–788.
- Teodorovic, D. and S. Guberinic. (1984). "Optimal Dispatching Strategy on an Airline Network After a Schedule Perturbation." *Journal of Operations Research* 15, 178–182.
- Teodorovic, D. and G. Stojkovic. (1990). "Model for Operational Daily Airline Scheduling." *Transportation Planning and Technology* 14, 273–285.
- Teodorovic, D. and G. Stojkovic. (1995). "Model to Reduce Airline Schedule Disturbances." *Journal of Transportation Engineering* 324–331.
- Thengvall, B., J. Bard, and G. Yu. (2000). "Balancing User Preferences for Aircraft Schedule Recovery During Irregular Operations." *IIE Transactions* 32(3), 181–193.
- Yan, S. and D.H. Yang. (1996). "A Decision Support Framework for Handling Schedule Perturbation." *Transportations Research Part B* 30(6), 405–419.
- Yan, S. and Y. Tu. (1997). "Multifleet Routing and Multistop Flight Scheduling for Schedule Perturbation." *European Journal of Operations Research* 103, 155–169.