Disruption Management and Planning with Uncertainties in Aircraft Planning

J. Ehrhoff, S. Grothklags, U. Lorenz

University of Paderborn, Germany

1 Introduction

An important problem in aircraft planning is to react with an instant decision, after a certain disruption which hinders the company to act as planned before. AI and OR have a long lasting history in the area of planning under uncertainty. For an introduction and a wider overview we refer e.g. to [16, 17].

Multistage Decisions under Risk The reason for disruptions obviously stems from the fact that planners lack information about the real behavior of the environment at planning time. Often, data is not as fixed as assumed in the traditional planning process. Instead we know the data approximately, we know distributions over the data. In our airline example, we know e.g. a distribution over a leg's (i.e. flight's) possible arrival times. Traditionally, plans are built which maximize profits over 'expected' or just estimated input data. We belong to the group of people who believe that it is more realistic to optimize the expected payoff over all possible scenarios instead. This view on the world leads us to something that is often called 'multistage decisions under risk', related to linear stochastic programming [4, 14], stochastic Optimization [10], game playing [3] and replanning [9].

Planning in Airline Industry

An airline planning process starts with the so called network design, which roughly tells the planning team which routes (so called *legs*) should be taken into account. Then, a first 'plan' is made which shows when which legs are offered to the customers. Thereafter, the planning process contains some layers which are of special interest for us.

Typically, airline companies have aircrafts of different types (so called *subfleets*), which differ in size and economic behavior. Given a flight schedule and a set of aircrafts, the fleet assignment problem is to determine which type of aircraft should fly each flight segment. A solution of the fleet assignment problem and the flight schedule together answers the question of how many aircrafts of which subfleet have to be at certain places at certain times. The fleet assignment problem is known to be NP-hard ([5]).

So called time-space networks, which are special flow graphs, can be used to give a specific mathematical programming formulation for this class of problems.

They were introduced by Hane et al. in [6] to solve the fleet assignment problem. On the basis of the fleet assignment, a so called rotation plan is generated. The rotation plan describes which physical aircraft must be at which place in the world and at which time.

The planning is dominated by deterministic models. All uncertainties are eliminated though restrictive models. However, since some time, several large airline companies have come to the opinion that new models and methods are necessary in order to exploit further potentials for cost reduction.

Fleet assignment and rotation planning belong to long- and midterm planning phases. They produce airline plans according to economical parameters (passenger demands, revenues, costs, ...), airline parameters (existing aircraft types, capacities, crews, ...) and operational restrictions (maintenance times, flight durations, ...). To enhance this profitability aircraft usage is more and more increased and airline plans become tighter and tighter. The tighter the plans become, the more frequently disruptions occur, which must be faced by the operation control management [15]. Disruptions cause delays, aircraft changes, cancellations, ad hoc crew re-assignment, slot problems, etc. Therefore, it is a major desire of the long- and midterm planning groups to build plans which allow the operation control to go back to the original plan fast and without causing high costs.

Game Tree Search Game tree search is the core of most attempts to make computers play games. The game tree acts as an error filter and examining the tree behaves similar to an approximation procedure.

Typically, a game playing program consists of three parts: a move generator, which computes all possible moves in a given position; an evaluation procedure which implements a human expert's knowledge about the value of a given position (these values are quite heuristic, fuzzy and limited) and a search algorithm, which organizes a forecast.

At some level of branching, the complete game tree (as defined by the rules of the game) is cut, the artificial leaves of the resulting subtree are evaluated with the heuristic evaluations, and these values are propagated to the root [8,13,1] of the game tree as if they were real ones. For 2–person zero-sum games, computing this heuristic minimax value is by far the most successful approach in computer games history, and when Shannon [18] proposed a design for a chess program in 1949 — which is in its core still used by all modern game playing programs — it seemed quite reasonable that deeper searches lead to better results. Indeed, the important observation over the last 40 years in the chess game and some other games is: *the game tree acts as an error filter*. Therefore, the faster and the more sophisticated the search algorithm, the better the search results! This, however, is not self-evident, as some theoretical analyzes show [2,12,7].

The evolution of game tree search in parlor games is important for us, because the computer games community has explicitly stated the question whether it is self-evident that more forecast leads to better results. Indeed, it is not self-evident. We miss this aspect of forecasting with scenarios in planning literature.

1.1 New Approach

Our approach can roughly be described by looking at a (stochastic) planning task in a 'tree-wise' manner. Let a tree T be given that represents the possible scenarios as well as our possible actions in the forecast time-funnel. It consists of two different kinds of nodes, MIN nodes and AVG nodes. A node can be seen as a 'system state' at a certain point of time at which several alternative actions can be performed/scenarios can happen. Outgoing edges from MIN nodes represent our possible actions, outgoing edges from AVG nodes represent the ability of Nature to act in various ways. Every path from the root to a leaf can then be seen as a possible solution of our planning task; our actions are defined by the edges we take at MIN nodes under the condition that Natures acts as described by the edges that lead out of AVG nodes.

The leaf values are supposed to be known and represent the total costs of the 'planning path' from the root to the leaf. The value of an inner MIN node is computed by taking the minimum of the values of its successors. The value of an inner AVG node is built by computing a weighted average of the values of its successor nodes. The weights correspond to realization probabilities of the scenarios.

Let a so called *min-strategy* S be a subtree of T which contains the root of T, and which contains exactly one successor at MIN nodes, and all successors that are in T at AVG nodes. Each strategy S shall have a value $f(S)$, defined as the value of S's root. A *principle variation* $p(S)$, also called *plan*, of such a min-strategy can be determined by taking the edges of S leaving the MIN nodes and a highest weighted outgoing edge of each AVG node. The connected path that contains the root is $p(S)$. We are interested in the plan $p(S_b)$ of the best strategy S_b and in the expected costs $E(S_b)$ of S_b. The expected costs $E(p)$ of a plan p are defined as the expected costs of the best strategy S belonging to plan p, e.g. $E(p) = \min\{E(S) \mid p(S) = p\}$. Because differences between planned operations and real operations cause costs, the expected costs associated with a given plan are not the same before and after the plan is distributed to customers. A plan gets a value of its own once it is published and other parties depend on it.

This model might be directly applied in some areas, as e.g. job shop scheduling [11], not, however, in applications which are sensible to temporary plan deviations. If a job shop scheduling can be led back to the original plan, the changes will nothing cost, as the makespan will stay as it was before. That is different in airline fleet assignments. Mostly, it is possible to find back to the original plan after some while, but nevertheless, costs occur. A decisive point will be to identify each tree nodes with a pair of the system state plus the path, how the state has been reached. At first glance this seems to complicate the given facts, but in truth this little detail enables our research to be practically relevant.

An analysis, as mentioned above, starts as soon as a disruption has occurred. We compute several partial plans which lead back to the original plan with low change-costs, examine further possible disruptions in the next time-step, compute further optional repair-plans, examine the next time-step etc., until we reach a search depth limited only by our computing power. The repaired plan that we select has small expected repair and future costs.

The appeal of our new approach, playing the so called *Repair Game*, lies in a slim problem description and in presenting solving techniques which provide the opportunity to plug in heuristics which quickly lead to good approximations, such that we are able to produce gains for large real world applications. The Repair Game represents a generic methodology for general logistic planning tasks to incorporate stochastic input data. The method has an AI search, OR and game theoretic flavor. For the sake of clarity, we restrict the description mainly to the area of airline scheduling. Before we start explaining our methodology for planning under uncertainty, we give a short introduction about the two other fields which influenced our work.

Our approach differs from known techniques in at least one of the following properties:

- We plan against the odds of an environment with the help of a local tree search into the future. When a disruption occurs, we compute several repair plans which lead back to the original plan, examine further possible disruptions in the next time-step, compute further repair-plans etc.
- We define robustness with the help of possible future events.
- We generate scenarios automatically and make profit from the bulk of scenarios. This bulk of scenarios acts like an error filter.
- The possible actions of the environment need not be equal in all branches of the forecast tree.
- We compare our experimental results with optimal results in the corresponding deterministic model.

References

1. A. de Bruin A. Plaat, J. Schaeffer and W. Pijls. A minimax Algorithm better than SSS*. *Artificial Intelligence*, 87:255–293, 1999.
2. I. Althöfer. Root evaluation errors: How they arise and propagate. *ICCA Journal*, 11(3):55–63, 1988.
3. B.W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.
4. S. Engell, A. Märkert, G. Sand, and R. Schultz. Production planning in a multiproduct batch plant under uncertainty. *Preprint 495-2001, FB Mathematik, Gerhard-Mercator-Universität Duisburg*, 2001.
5. Z. Gu, E.L. Johnson, G.L. Nemhauser, and Y. Wang. Some properties of the fleet assignment problem. *Operations Research Letters*, 15:59–71, 1994.
6. C.A. Hane, C. Barnhart, E.L. Johnson, R.E. Marsten, G.L. Nemhauser, and G. Sigismondi. The fleet assignment problem: solving a large-scale integer program. *Mathematical Programming*, 70:211–232, 1995.
7. H. Kaindl and A. Scheucher. The reason for the benefits of minmax search. In *Proc. of the 11 th IJCAI*, pages 322–327, Detroit, MI, 1989.
8. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

9. S. Koenig, D. Furcy, and Colin Bauer. Heuristic search-based replanning. *In Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*, pages 294–301, 2002.
10. P. Kouvelis, R.L. Daniels, and G. Vairaktarakis. Robust scheduling of a two-machine flow shop with uncertain processing times. *IIE Transactions*, 32(5):421–432, 2000.
11. V.J. Leon, S.D. Wu, and R.h. Storer. A game-theoretic control approach for job shops in the presence of disruptions. *International Journal of Production Research*, 32(6):1451–1476, 1994.
12. D.S. Nau. Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence*, 21(1-2):221–244, 1983.
13. A. Reinefeld. An Improvement of the Scout Tree Search Algorithm. *ICCA Journal*, 6(4):4–14, 1983.
14. W. Römisch and R. Schultz. Multistage stochastic integer programming: an introduction. *Online Optimization of Large Scale Systems*, pages 581–600, 2001.
15. J. M. Rosenberger. *Topics in Airline Operations*. 2000. PhD-Thesis, Georgia Institute of Technology, Atlanta, GA 30332.
16. S. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach*. 2003. Prentice Hall Series in Artificial Intelligence.
17. A. Scholl. Robuste Planung und Optimierung. *Physiker Verlag*, 2001.
18. C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.