Parallelism for Perturbation Management and Robust Plans*

Jan Ehrhoff[2], Sven Grothklags[1], and Ulf Lorenz[1]

[1] University of Paderborn
Faculty of Computer Science, Electrical Engineering and Mathematics
Fürstenallee 11, D-33102 Paderborn
[2] Lufthansa Systems Airline Services GmbH
Network Management Solutions
Am Prime Parc 9, D-65479 Raunheim

Abstract. An important insufficiency of modern industrial plans is their lack of robustness. Disruptions prevent companies from operating as planned before and induce high costs for trouble shooting. The main reason for the severe impact of disruptions stems from the fact that planners do traditionally consider the precise input to be available at planning time.
The Repair Game is a formalization of a planning task, and playing it performs disruption management and generates robust plans with the help of game tree search. Technically, at each node of a search tree, a traditional optimization problem is solved such that large parts of the computation time are blocked by sequential computations. Nevertheless, there is enough node parallelism which we can make use of, in order to bring the running times onto a real-time level, and in order to increase the solution quality per minute significantly. Thus, we are able to present a planning application at the cutting-edge of Operations Research, heavily taking advantage of parallel game tree search. We present simulation experiments which show the benefits of the repair game, as well as speedup results.

1 Introduction

An important problem in aircraft planning is to react with an instant decision, after a certain disruption hinders the company to act as planned before. This problem touches various research directions and communities. Because the problems are often computationally hard [14] the field might become an El Dorado for parallel computing and grid computing. A stochastic multi-stage fleet-assignment optimization problem is in the focus of this paper. The used solution method is based on game tree search.

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

* This work has been partially supported by the European Union within the 6th Framework Program under contract 001907 (DELIS) and the German Science Foundation (DFG), SFB 614 (Selbstoptimierende Systeme des Maschinenbaus) and SPP 1126 (Algorithmik großer und komplexer Netzwerke).

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 1265–1274, 2005.

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 the airline example, we know 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, but 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, 16], stochastic Optimization [11], game playing [2], replanning [10] and others [18].

Current Planning Processes 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 process contains two 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.

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 [7] to solve the fleet assignment problem. On the basis of the fleet assignment, a so called rotation plan is generated. It 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 through restrictive models. However, since some time, several large airline companies have come to the conclusion that new models and methods are necessary in order to exploit further potentials for cost reduction.

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. 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 help of heuristics, and these values are propagated to the root [9, 15] 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 [19] proposed a design for a chess program in 1949 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 [1, 8, 13].

New Approach. Our approach [3] 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.

This model might be directly applied in some areas, as e.g. job shop scheduling [12], 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.

1.1 Organization of This Paper

We introduce the Repair Game as a reasonable formalization of the airline planning task on the level of disruption fighting. Section 2 describes the Repair Game, its formal definition, as well as an interpretation of the definition and an example. In Section 3 we describe a prototype, which produces robust repair decisions for disrupted airline schedules, on the basis of the Repair Game. Section 4 contains details of the parallelization of the search procedure. In Section 5 we compare the results of our new approach with an optimal repair procedure (in the traditional sense). A comparison of the sequential and the parallel version of our prototype is additionally given. Section 6 concludes.

2 The Repair Game

Definitions. We define the Repair Game via its game tree. Its examination gives us a measure for the robustness of a plan and on the other hand it presents us concrete operation recommendations.

Definition 1. *(Game Tree)*
For a rooted tree $T = (V, E)$ let $L(T) \subset V$ be the set of leafs of T. In this paper, a game tree *$G = (V, E, h)$ is a rooted tree (V,E), where $V = V_{MAX} \cup V_{MIN} \cup V_{AVG}$ and $h : V \to \mathbb{N}_0$.*

Nodes of a game tree G represent positions of the underlying game, and edges move from one position to the next. The classes V_{MAX}, V_{MIN}, and V_{AVG} represent three players MAX, MIN, and AVG and for a node/position $v \in V_i$ the class V_i determines the player i who must perform the next move.

Definition 2. *(*Minimax *Value)*
Let $G = (V, E, h)$ be a game tree and $w_v : N(v) \to [0, 1]$ be weight functions for all $v \in V_{AVG}$, where $N(v)$ is the set of all sons of a node v. The function *minimax $: V \to \mathbb{N}_0$ is inductively defined by*

$$*\text{minimax}(v) := \begin{cases} h(v) & \text{if } v \in L(G) \\ \max\{*\text{minimax}(v') \mid v' \in N(v)\} & \text{if } v \in V_{MAX} \setminus L(G) \\ \min\{*\text{minimax}(v') \mid v' \in N(v)\} & \text{if } v \in V_{MIN} \setminus L(G) \\ \sum_{v' \in N(v)} (w_v(v') \cdot *\text{minimax}(v')) & \text{if } v \in V_{AVG} \setminus L(G) \end{cases}$$

Definition 3. *(Repair Game)*
The goal of the Repair Game *= (G,p,g,f,s) is the calculation of* *minimax(r) *for a special game tree $G = (V, E, g + f)$ with root r and uniform depth t; $p \in L(G)$ is a special leaf, g, f and s are functions. The game tree has the following properties:*

- *Let $P = (r = v_1, v_2, \ldots, p = v_t) \in V^t$ be the unique path from r to p. P describes a traditional, original plan.*
- *V is partitioned into sets $S_1, \ldots, S_n, |V| \geq n \geq t$ by the function $s : V \to \{S_i\}_{1 \leq i \leq n}$. All nodes which belong to the same set S_i are in the same* state of the system *— e.g. in aircraft scheduling: which aircraft is where at which point of time —, but they differ in the histories which have led them into this state.*
- *$g : \{S_i\}_{1 \leq i \leq n} \to \mathbb{N}_0$ defines the expected future costs for nodes depending on their state; for the special leaf p holds g(s(p)) = 0*
- *$f : \bigcup_{1 \leq \tau \leq t} \{V\}^\tau \to \mathbb{N}_0$ defines the induced repair-costs for every possible (sub)path in (V, E) ; every sub-path P' of P has zero repair-costs, f(P') = 0*
- *the node evaluation function $h : V \to \mathbb{N}_0$ is defined by $h(v) = g(s(v)) + f(r \ldots v)$; note that $h(p) = 0$ holds by the definition of g and f*

2.1 Interpretation and Airline Example

A planning team of e.g. an airline company starts the game with the construction of a traditional plan for its activities. The path P represents this planned schedule, which

also is the most expected path in the time-funnel, and which interestingly gets an additional value of its own, as soon as it is generated. It is small, can be communicated, and as soon as a customer or a supplier has received the plan, each change of the plan means extra costs for the change. Disruptions in airline transportation systems can now prevent airlines from executing their schedules as planned. As soon as a specific disruption occurs, the MIN-player will select a repairing sub-plan such that the repair costs plus the expected future repair costs are minimized.

As the value of a game tree leaf v depends on how 'far' the path $(r, \ldots, v)$ is away from P, it will not be possible to identify system states (where the aircrafts are at a specific time) with tree nodes. Therefore, the tree nodes V are partitioned into $S_1 \cup \ldots \cup S_n$. In S_i all those nodes are collected which belong to the same state, but have different histories. All nodes in the same state S_i have the same expected future costs. These costs are estimated by the function g. The function f evaluates for an arbitrary partial path, how far it is away from the level-corresponding partial path of P. Inner nodes of the game tree are evaluated by the *Minimax function.

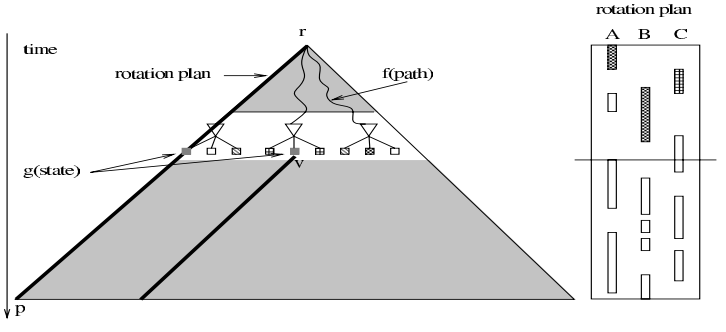

Fig. 1. The Repair Game Tree.

Figure 1 shows a rotation plan at the right. Aircrafts A, B, and C are either on ground, or in the air, which is indicated by boxes. A shadowed box means that the original plan has been changed. The time goes from the top down. The left part of the figure shows a game tree, where the leftmost path corresponds to the original plan P. When a disruption occurs, we are forced to leave the plan, but hopefully we can return to it at some node v. The fat path from node v downward is the same as in the original plan. Thus, at node v, we have costs for the path from the root r to v, denoted by $f(r \ldots v)$ and future expected costs, denoted by $g(s(v))$. If we follow the original plan from the beginning on, we will have the same expected costs at the time point represented by v, but different path costs. The only node with zero costs typically is the special leaf node p of the original plan.

3 Experimental Setup

In accordance with our industrial partner, we built a simulator in order to evaluate different models and algorithms The simulator is less detailed than e.g. SimAir [17], but we

believe that it is detailed enough to model the desired part of the reality. Furthermore, it is simple enough such that the occurring problems are computationally solvable.

First of all, we discretize the time of the rotation plan into steps of $d = 15$ minutes. Every departure in the current rotation plan is a possible event point, and all events inside one d minute period are interpreted as simultaneous. When an event occurs, the leg which belongs to that event can be disrupted, i.e. it can be delayed by 30, 60, or 120 minutes, or it can be canceled with certain probabilities. Let T be the present point of time. The simulator inspects the events between T and $T + d$, and informs a repair engine about the new disruptions, waits for a corrected rotation plan, and steps d minutes forward.

The aim is to compare two different repair approaches with each other. The first one is called 'Myopic MIP' solver and it repairs a plan after a disruption with the help of a slightly modified time-space network such that the solution of the associated network flow problem is an optimal one, under the assumption that no more disruptions will ever occur. This engine represents traditional deterministic planning. It only needs a modified cost function because the repair costs are mainly determined by the changes on the original plan, rather than by leg profits.

The second engine, called 'T3', is an engine which plays the Repair Game. It compares various solutions of the modified time-space network flow problem and examines various scenarios which might occur in the near future. The forecast procedure makes use of the dynamics time-locally around time T and $T + d$ as follows: Instead of generating only one myopic MIP optimal solution for the recovery, we generate several ones. They are called our *possible moves*. A simple, certainly good heuristic is to demand that these solutions have nearly optimal costs concerning the cost function which minimizes the cost for changes. For all of these possible moves, we inspect what kind of relevant disruptions can come within the next d minutes. On all these scenarios, we repair the plan again with the help of the myopic MIP solver, which gives us value estimations for the arisen scenarios. We weight the scenarios according to their probabilities, and minimize over the expected values of the scenarios. Concerning the new part of the plan we have not optimized the costs over expected input data, but we have approximately minimized the expected costs over possible scenarios. The following algorithm is a simplified version of the algorithm shown in [2]. We refer to [3] for further details of the repair engines.

value *minimax(node v, value α, value β)
1 generate all successors $v_1, \ldots, v_b$ of v; let b be the number of successors
2 value $val := 0$;
3 **if** $b = 0$ return $h(v)$ /* (leaf eval)*/
4 **for** $i := 1$ **to** b
5 **if** v is MIN-node {
6 $\beta :=$ min(β, *minimax(v_i, α, β)); **if** $\alpha \geq \beta$ return β; **if** $i = b$ return β
7 } **else** **if** v is AVG-node { // let $w_1, .., w_b$ be the weights of $v_1, .., v_b$
8 $val+ =$ *minimax(v_i, α, β) $\cdot\, w_i$;
9 **if** $val + L \cdot \sum_{j=i+1}^{b} w_j \geq \beta$ return β; **if** $val + U \cdot \sum_{j=i+1}^{b} w_j \leq \alpha$ return α
10 **if** $i = b$ return val
11 } **else** { // v is MAX-node. Analogously to v is a MIN-node }

4 Parallelization

The basic idea of our parallelization is to use dynamic load balancing and to decompose the search tree, to search parts of the search tree in parallel and to balance the load dynamically with the help of the work stealing concept. This works as follows: First, a special processor P_0 gets the search problem and starts performing the *minimax algorithm as if it would act sequentially. At the same time, the other processors send requests for work, the REQUEST message, to other randomly chosen processors. When a processor P_i that is already supplied with work, catches such a request, it checks, whether or not there are unexplored parts of its search tree, ready for evaluation. These unexplored parts are all rooted at the right siblings of the nodes of $P_i's$ search stack. Either, P_i sends back that it cannot serve with work with the help of the NO-WORK message, or it sends such a node (a position in the search tree etc.) to the requesting processor P_j. That is done with the help of the WORK message. Thus, P_i becomes a master itself, and P_j starts a sequential search on its own. The master/worker relationships are dynamically changed during the computation. A processor P_j being a worker of Q at a certain point of time may become the master of Q at another point of time. In general, processors are masters and workers simultaneously. If, however, a processor P_j has evaluated a node v, but a sibling of v is still under examination of another processor Q, P_j will wait until Q sends an answer message. When P_j has finished its work (possibly with the help of other processors), it sends an ANSWER message to P_i. The master-worker relationship between P_i and P_j is released, and P_j is idle again. It again starts sending requests for work into the network.

When a processor P_i finds out that it has sent a wrong local bound to one of its workers P_j, it makes a WINDOW message follow to P_j. P_j stops its search, corrects the window and starts its old search from the beginning. If the message contained a cutoff, P_j just stops its work. A processor P_i can shorten its search stack due to an external message, when e.g. a CUTOFF message comes in which belongs to a node near the root. In absence of deep searches and advanced cutting techniques, however, CUTOFF and WINDOW messages did not occur in our experiments.

In many applications, the computation at each node is fast in relation to the exchange of a message. E.g. there are chess programs which do not use more than 1000 processor cycles per search node, including move generation, moving end evaluating the node. In the application presented here, the situation is different. The sequential computations at each node takes several seconds if not even minutes, such that the latency of messages is not remarkably important. Therefore, the presented application is certainly well suited for grid computing, as well. However, it is necessary to overlap communication and computations. We decoupled the MIP solver from the rest of the application and assigned a thread to it.

In distributed systems, messages can be delayed by the system. Messages from the past might arrive, which are outdated. Therefore, for every node v a processor generates a local unique ID, which is added to every message belonging to node v. Thus, we are always able to identify misleading results and to discard invalid messages. We refer to [5] for further details.

Table 1. The daily-average profit of T3 over Myopic, two different weeks, six measure points each.

	run 1	run 2	run 3	run 4	run 5	run 6
week 1	2320	27696	32261	-9238	-15799	13150
week 2	11040	48778	11580	-1253	9144	8389

5 Results

The basis for our simulations is a major airline plan plus the data which we need to build the plan and its repairs. The plan consists of 20603 legs, operated by 144 aircrafts within 6 different subfleets. The MIP for this plan has 220460 columns, 99765 rows, and 580793 non-zeros. Partitioned into single days, the resulting partial plan for any single day consists of a corresponding smaller number of columns and non-zeros.

All experiments are performed on a 4-node-cluster of the Paderborn University. Each node consists of two Pentium IV/2.8 GHz processors on a dual processor board with 1 GB of main memory. The nodes are connected by a Myrinet high speed interconnection network. The cluster runs under the RedHat Linux operating system.

We simulated 14 days, and we divided this time period into 14 pieces such that we arrived at a test set with 14 instances. Moreover, time is divided into segments of 15 minutes, and everything happening in one 15 minute block is assumed to be simultaneous. We compare the behavior of the 'myopic MIP' and the 'T3' engines. We appropriately choose the probabilities for disruptions and control the performance of the objective function $c(TIM, ECH, CNL, revenue) = 50 \cdot TIM + 10000 \cdot ECH + 100000 \cdot CNL - revenue$, TIM being time-shifts, ECH meaning that the aircraft type of a leg had to be changed, CNL being the number of necessary cancellations and revenues being the traditional cost measure for the deterministic problem. A test run consists of about 1400 single decisions, and first tests showed benefits of more than three percent cost reductions over the myopic solver. Although a benefit of more than three percent looks already promising, statistical significance cannot be read out of a single test run. The engines make nearly 100 decisions per simulated day. However, these decisions cannot be taken as a basis for significance examinations, because the single decisions are part of a chain and not independent from each other. The results of the single days seem to form no normal distribution and, moreover, depend on the structure of the original plan. Therefore, we made further test runs and grouped those outcomes of each simulated week to one. We measure the average daily profit of a week, in absolute numbers (see Table 1). The profit, which is statistically significant, is the benefit of the T3-engine over the Myopic-MIP engine.

5.1 Speedups

We measure speedups of our program with the help the first three days of the test set which consists of the 14 single days, mentioned above. The time for simulation of the days using one processor is compared with the running time of several processors. The speedup (SPE) is the sum of the times of the sequential version divided by the

Table 2. Speedups.

# proc	simtime day 1	SPE day1	simtime day 2	SPE day2	simtime day 3	SPE day3
1	226057	1	193933	1	219039	1
2	128608	1.76	111612	1.73	126915	1.73
4	68229	3.31	59987	3.23	66281	3.30
8	46675	4.84	40564	4.78	46065	4.75

sum of the times of a parallel version [6]. Each test day consists of 96 single measure points. Table 2 shows the speedups which we could achieve. We are quite satisfied with these speedups because they bring the necessary computations on a real-time level. Of course, the question arises how the work load and the search overhead behave. As the sequential and the parallel version do indeed exactly the same, search overhead does not exist. Neither the costs for communication are relevant. The only reason that we cannot achieve the full speedup are the large sequential computing periods, caused by the commercial MIP solver.

6 Conclusion

Playing the Repair Game leads to more robust (sub-)plans in airline scheduling than traditional deterministic planning can provide. Our forecast strategy outperforms the myopic MIP solver by means of simulations. We have parallelized the search in order to drop the computation times to real time. Next, we will look for more clever and selective search heuristics, examine heuristics which give us fast new moves, and refine the simulator.

We presented an application which we think is a typical example for the benefits of cluster parallelism and grid computing. The stochastic fleet assignment problem that we presented in the frame of game tree search makes profit from its speed. The faster the application can be performed, the larger is the gained profit. Planning under uncertainty becomes more and more important in Operations Research. The resulting problems are hard to solve and can often only be approximated. We are convinced that parallel game tree search will become an important part of that area.

References

1. I. Althöfer. Root evaluation errors: How they arise and propagate. *ICCA Journal*, 11(3):55–63, 1988.
2. B.W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.
3. J. Ehrhoff, S. Grothklags, and U. Lorenz. Das Reparaturspiel als Formalisierung von Planung unter Zufallseinflüssen, angewendet in der Flugplanung. In *Proceedings of GOR conference: Entscheidungsunterstützende Systeme in Supply Chain Managment und Logistik*, pages 335–356. Physika-Verlag, 2005.
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. R. Feldmann, M. Mysliwietz, and B. Monien. Studying overheads in massively parallel min/max-tree evaluation. In *6th ACM Annual symposium on parallel algorithms and architectures (SPAA'94)*, pages 94–104, New York, NY, 1994. ACM.
6. P.J. Flemming and J.J. Wallace. How not to lie with statistics: the correct way to summerize benchmark results. *CACM*, 29(3):218–221, 1986.
7. 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.
8. 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.
9. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
10. 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.
11. 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.
12. 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.
13. D.S. Nau. Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence*, 21(1-2):221–244, 1983.
14. C. H. Papadimitriou. Games against nature. *Journal of Computer and System Science*, 31:288–301, 1985.
15. A. Reinefeld. An Improvement of the Scout Tree Search Algorithm. *ICCA Journal*, 6(4):4–14, 1983.
16. W. Römisch and R. Schultz. Multistage stochastic integer programming: an introduction. *Online Optimization of Large Scale Systems*, pages 581–600, 2001.
17. J. M. Rosenberger, A. J. Schaefer, D. Goldsman, E. L. Johnson, A. J. Kleywegt, and G. L. Nemhauser. Simair: A stochastic model of airline operations. *Winter Simulation Conference Proceedings*, 2000.
18. S. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach*. 2003. Prentice Hall Series in Artificial Intelligence.
19. C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.