A global approach to crew-pairing optimization

by R. Anbil
R. Tanga
E. L. Johnson

The problem addressed in this paper is crew-pairing optimization in airline flight planning: finding tours of duty (pairings) that are legal and cover every flight leg at the least cost. The legal rules and cost of a pairing are determined by complex Federal Aviation Agency and contractual requirements. In addition, the problem is made more difficult by the hub-and-spoke system used by airlines that multiplies the possible ways a pairing can link flight legs. The state-of-the-art crew-pairing TRIP system of American Airlines uses subproblem optimization and, as is true for other crew-scheduling systems, may not be able to improve a solution even though a better one exists. We report on the methodology developed during a joint study by IBM and American Airlines Decision Technologies to use the IBM Optimization Subroutine Library in conjunction with TRIP to improve on crew-pairing solutions by taking a global approach. The resulting improvements have been a reduction of 5 to 11 percent in excess crew cost. Estimated total savings are five million dollars per year.

American Airlines (AA) employs more than 25 000 pilots and flight attendants to fly its fleet of over 600 aircraft. Crew cost is over 1.3 billion dollars per year and is second only to fuel cost. In its effort to better utilize crew resources, AA spent about 6000 hours of CPU time on an IBM 3090* system during 1989–90, running its crew-pairing code, the Trip Reevaluation and Improvement Program (TRIP). Estimated savings generated by the improvements in the TRIP code during the past five years are in excess of 20 million dollars per year.[1]

A crew pairing is a sequence of flights that starts and ends at a crew base and typically lasts two or three days. A crew member works four or five pairings per month. The most important part of efficient crew utilization is making up pairings that cover all flight legs and minimize excess cost. This problem is called crew-pairing optimization. Figure 1 illustrates a pairing that begins and ends at the Dallas–Fort Worth (DFW) Airport, an AA crew base.

One of the difficulties in constructing pairings is observing the many union and Federal Aviation Agency (FAA) rules governing the legality and penalties of a pairing. Some of these rules have to do with the duration and flying time of a duty period, i.e., the time that a crew is flying or is between flights, but not resting overnight. During the planning stages, a duty period is usually restricted to eight hours of flying and 12 hours of total duty time, including briefing and debriefing. Between duty periods, there are overnight rests or layovers that must exceed some minimum duration. The most difficult legal rule to check concerns the longer layover time required if there have been more than eight hours of flying in any 24-hour period preceding the layover.

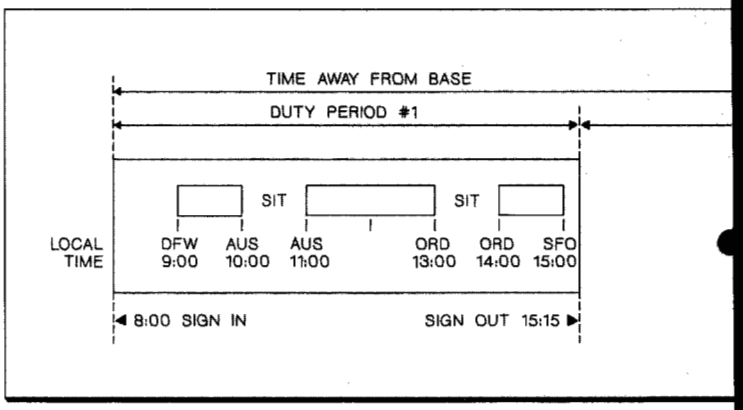
In addition to legal rules for pairings, the cost structure is also very complex. The main components of excess cost are pay and credit: guaranteed hours of pay minus actual hours flown. The three main causes of pay and credit are pairings that include (1) long or frequent sits within a duty period, (2) long overnight rests between duty periods, and (3) "deadheading" (transporting crews as passengers). Pay and credit are calculated as a maximum of several pay guarantees.

There are three considerations other than legality and excess costs that a crew-pairing optimizer must take into account. First, crew assignments must consider the number of crews available at the various crew bases. This constraint is called crew balance and is usually expressed as a minimum and maximum number of flying hours per month available at each crew base. The second consideration is preference for keeping crews on the same plane during a duty period as much as possible. In TRIP, a penalty is added to the cost of a pairing for each time the crew changes planes during a duty period. The third consideration is preference for shorter pairings. Usually, AA restricts the maximum lengths of domestic pairings to three days, i.e., three duty periods. The restriction is imposed because longer pairings would cause greater difficulty in rescheduling if weather or other factors cause a pairing to be disrupted. In addition, computational experience has shown that longer pairings do not generally lead to significantly lower costs.

In addition to the complexity of the rules and checks for legality, another reason for the inherent difficulty of this problem is the explosion in the number of pairings caused by the hub-and-spoke network of AA's domestic schedule. As mentioned before, it is preferable to leave the crew on the same plane as long as possible. However, a change of planes for the crew at a hub can definitely lead to better pairings, and allowing this change of planes explodes the number of possible pairings. Just as passengers can connect in many ways, so can the crews.

Crew-pairing optimization is normally a monthly planning problem. This frequency is caused by monthly flight schedule changes. When there are mid-month adjustments to the flight schedule, the crew-pairing problem must be readdressed during the month. In addition, the transition period from the end of one month to the beginning of the next month poses special scheduling difficulties and frequently forces some deadheading in order to get crews home.

Figure 1 Example of crew pairing with DFW airport as crew base

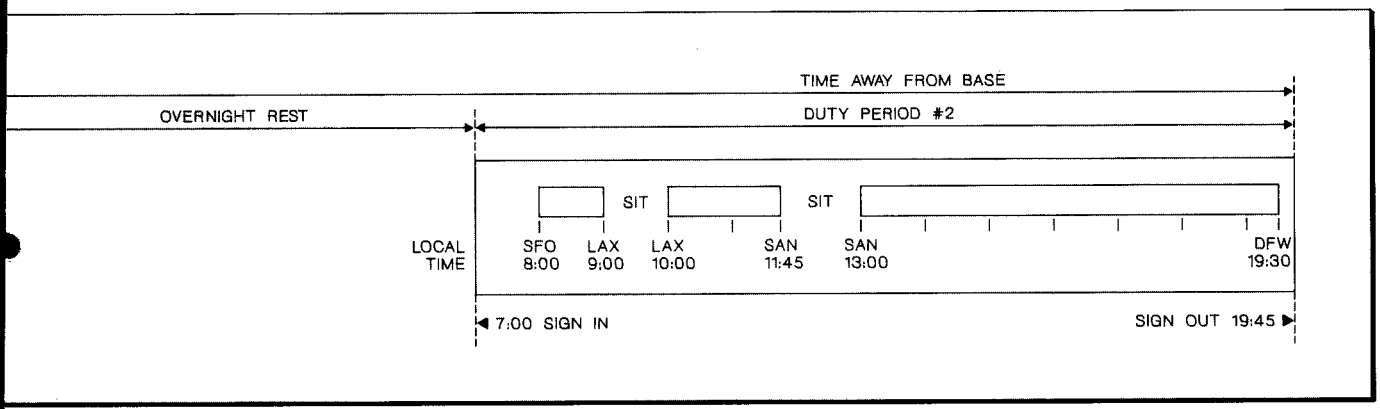


Thus, crew-pairing optimization is a particularly challenging combinatorial optimization problem. The next section discusses current methodology to address this intriguing problem.

Current methodology

AA's crew-pairing optimization system, TRIP, has been continuously improved over the last 20 years, with major improvements in the last three years.[1] Despite these improvements it continues to be based on a subproblem approach that may lead to suboptimum solutions. In this section, the subproblem approach is explained. The following sections present a new approach jointly developed by American Airlines Decision Technologies (AADT) and IBM that partially overcomes those shortcomings.

The overall crew-pairing optimization process consists of sequentially solving several problems. The first problem solved is called the daily problem. It assumes that the flight segments are flown every day. The advantage of looking at the daily problem is twofold. First, the problem becomes more tractable because the daily problem only has a number of segments to cover that are equal to the number of daily segments, i.e., any pairing that includes a segment in any of its duty periods will cover that segment every day, since the same pairing will be started each day. The largest number of daily segments in AA's schedule is about



one thousand, even for its largest fleet. The second advantage of the daily pairing problem is that, from an operational standpoint, it gives regularity to crew assignments.

After daily pairings have been determined from the solution to the daily problem, there are several other phases to the crew-pairing optimization process. Crew base constraints must be met, weekly exceptions must be handled, and the transition period must be planned. The subsequent discussion deals with the daily problem, which is the heart of the crew-pairing optimization problem.

The daily problem solution procedure begins by finding a workable solution. This initial phase uses a code called the Initial Solution Generator and attempts to adapt the daily solution of the previous months to the current month. The TRIP code then iteratively selects and solves a subproblem in order to improve the initial solution. Each TRIP iteration consists of three phases. The first phase is subproblem selection. This process starts by choosing some number of pairings, typically 5 to 10, from all of the pairings that cover the daily flight segments. The subproblem then consists of the segments covered by the chosen set of pairings. For this smaller number of segments, all possible pairings are generated. Thus, pairing generation is the second phase. The pairings are checked for legality based on all of the rules. Additionally, the cost is calculated for each pairing, including all applicable penalties. The third phase of each TRIP iteration is the optimization phase. If there is a better set of pairings that exactly covers these chosen segments, the new pairings replace the chosen set of old pairings. In any case, we are ready to begin the next iteration.

The optimization problem is formulated as a set-partitioning problem.[2] This well-known integer programming problem has a coefficient matrix with a row for each flight segment and a column for each pairing. The entries are 0 or 1, with a 1 signifying that the pairing for the column includes the segment corresponding to that row. The next section gives the precise formulation of the set-partitioning problem. Details about the optimization method used in TRIP are available in Reference 1.

As is true for any subproblem approach, TRIP may reach a solution that cannot be improved even after many additional hours of CPU time have been expended doing further iterations. The reason is that even if each subproblem is optimized, an improvement requiring large changes to the incumbent solution cannot be found when the subproblems are each too small. The subproblem optimizer cannot handle large problems and is effectively limited to problems of about 100 segments and 10 000 columns. The main thrust of our efforts is to overcome this deficiency in the subproblem approach. This paper describes a global approach that, although falling short of global optimization, takes into consideration the entire problem and constructs a crew-pairing solution without using the initial solution.

Global approach and SPRINT code

Many millions of pairings are first generated. A corresponding linear program can be constructed by creating a column for each pairing and a row for each segment. The constraints require that each segment be covered exactly once. The cost of the variable associated with a pairing is the total excess cost of the pairing, including penalties. The resulting integer program is called a set-partitioning problem and has the form:

$$x_j = 0 \text{ or } 1, j = 1, \cdots, n$$

$$\sum_{j=1}^{n} a_{ij} x_j = 1, \text{ all } i = 1, \cdots, m$$

$$\text{minimize } \Sigma\, i_j x_j$$

where m is the number of segments, n is the number of pairings, and

$$a_{ij} = \begin{cases} 1 \text{ if pairing } j \text{ includes segment } i \\ 0 \text{ otherwise} \end{cases}$$

c_j = total excess cost of pairing j

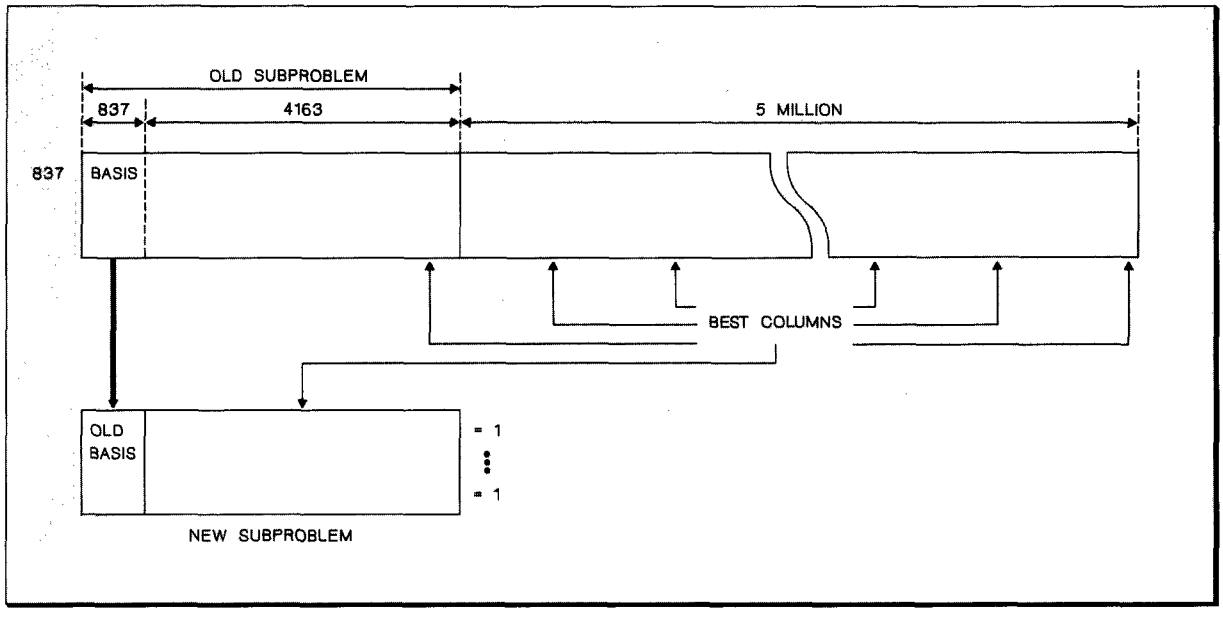
The linear programming relaxation relaxes $x_j = 0$ or 1 to $0 \leq x_j \leq 1$ and thus allows fractional values of the variables so that a given segment may be covered by fractional values of two or more pairings. For that reason, the solution is unusable in practice. Moreover, this set-partitioning linear program is widely known to be a class of difficult linear programs to solve. The benefits derived from solving the linear program are discussed in the next section. The rest of this section will explain the method used to solve set-partitioning linear programs having a very large number of columns.

The IBM–AADT joint study began by generating 12 million pairings, and these pairings were converted to input for the IBM Optimization Subroutine Library (OSL). This problem presented a double challenge in that even small set-partitioning linear programming (LP) problems may be difficult to solve, and this problem had 12 million columns. Removing duplicate columns and keeping the ones with lowest costs among the duplicates reduced the problem to 5.5 million columns (5 534 503 to be exact). During the summer of 1989, this LP problem was solved to optimality at the IBM Thomas J. Watson Research Center on a 3090 Vector Facility. Anuj Mehrotra, a summer student, implemented and tested several iterative approaches. The first attempts involved solving ever-increasingly larger sets of columns. The 5.5-million-column problem was never solved in this way.

The method that did work was the "SPRINT" approach of John Forrest. This method requires solving a subproblem consisting of a small subset of the columns and using the optimum dual variables from the subproblems to price out all 5.5 million columns. A new subproblem is formed by retaining only the columns in the optimum basis of the old subproblem and collecting some small set of good columns based on the new reduced costs. A good number of columns for the subproblems proved to be 5000. The idea of choosing more than one column that prices out to enter the basis is an old idea. What is different here is that 5000 such columns were collected. The main result, however, is an empirical one: this method works well for set-partitioning linear programs with a huge number of columns. The critical factor was to price out over all of the 5.5 million columns every time a new subproblem was created. The initial CPU time on a 3090E Vector Facility was about 12 hours. Forrest then worked on the SPRINT code and reduced the running time to 56.9 minutes. He tuned the subproblem size and selection process. The idea in column selection is to collect columns into "buckets" based on reduced costs and to fill up the required subproblem columns based on the best buckets first, until the required number of columns would be exceeded by a bucket. At this point, columns are chosen randomly from the last bucket. Also, to improve performance, Forrest perturbed significantly the right side of the problem in order to decrease degeneracy. Using Forrest's SPRINT code, many problems of 2 to 12 million columns were solved on the 3090 Vector Facility at IBM's National Engineering/Scientific Support Center in Dallas, Texas, and solution times were consistently less than one hour. When a good starting basis was available, running times to reach optimality were typically 20 minutes. Currently the RISC System/6000* Model 540 is being used with good success at AADT to solve 2- to 3- million-column problems. The SPRINT approach is illustrated in Figure 2.

The good performance of the SPRINT code is due in part to the remarkably small number of sub-

Figure 2 SPRINT approach to solving the 5.5-million-column problem



problems that need to be solved to globally optimize the large problems. In the case of the 5.5-million-column problem, only 25 subproblems were solved. One way to look at the solution process is to note that at most 125 000 (= 25 × 5000) columns out of 5.5 million ever got into any of the small LP problems.

Another reason for the small solution time is that the 5000-column problems could be solved relatively quickly. Even these problems are difficult linear programs, being of the set-partitioning type for which degeneracy is known to lead to poor performance, particularly for primal simplex codes. We do use primal simplex, and degeneracy is a problem, but the SPRINT OSL code has enough tools available to provide good performance. Primal, rather than dual, simplex was used because for each subproblem the optimum basis for the previous problem is available and is feasible from a primal, but not dual, standpoint.

One surprise that we found from the solution of the 5.5-million-column problem was that it was not as massively degenerate as is usually the case for set-partitioning problems. Degeneracy can be measured by the percent of basic variables at or near zero in the basic solution. For typical smaller set-partitioning problems, 80 percent degeneracy is common. For the 5.5-million-column problem it was only about 25 percent degenerate. That is, out of 837 basis variables, over 600 were positive in the linear programming optimum basic solution. The linear programming problems are still difficult; in fact we found the degenerate problems in the 25 percent category to be more difficult than the ones in the 80 percent category. The less-degenerate problem was typically associated with a better linear programming objective value and usually with a larger difference between the linear programming objective value and the objective value of the initial integer solution.

In fact, the degree of degeneracy usually corresponds to the degree of integrality of the linear programming solution. Folklore has it that linear programming solutions to set-partitioning problems are usually integer or close to integer. Our experience for small crew-scheduling set-partitioning problems bears out that contention. However, the problems that involve millions of columns have linear programming solutions that are consistently far from integer. The original 5.5-million-column problems had one column in the linear programming solution at a value of one. The values of the rest of the variables that were positive in the linear programming optimum solution were spread out between zero and one.

The real interest raised by the linear programming solution was not its countering of folklore but its objective function value. The original best solution from TRIP was about $57,000 per day, and the LP solution was about $48,000. That value was later lowered to almost $46,000 using some more columns. The fact that the linear programming objective value could be improved from $48,000 to $46,000 by generating more columns in addition to the original 12-million columns, points out that there are really billions of columns and no one knows how small the optimum linear programming objective function value might be. However, from a practical point of view it was already good enough: about $11,000 per day. The more pressing question was how much of that could be realized by an integer solution. The next section describes methodology for finding an integer solution. That problem turned out to be much more difficult than anticipated.

Integer solutions

The problem of finding integer solutions was much more difficult than originally envisioned. For smaller crew-scheduling problems, simply rounding up variables closest to one and resolving the linear programming worked well. However, that approach failed on the large problem because the resulting smaller problems eventually became infeasible. The usual branch-and-bound codes did not work well either, even on small subproblems. As a result, a special-purpose code was written that exploits the connectivity of the flight schedule to "lock" connecting segments. When every segment has been linked to a unique connecting segment, we have found a feasible integer solution, and we terminate.

The branching scheme that worked best was one suggested by Ryan.[3] The idea is to look at each segment and at every pairing with positive solution values in the optimum linear programming. In general, any given segment will have some number of segments following it in the various pairings. Although the pairings may have fractional linear programming values, it may be that every pairing involving the given segment has the same follow-on segment. In this case, we fix that follow-on; otherwise, the follow-on is fixed to be the segment that has the largest LP value among all of the follow-ons for all of the segments. In essence, we are letting the LP solution decide on a most favorable connecting segment to some segment. Then, that connection is fixed from there on. Ryan suggested branching on follow-ons, and since we only explore one of the "branches," our implementation is not done by branching. For that reason, we chose the one that we expect to be correct.

The overall scheme is to first get a good linear programming solution as described in the previous section. Then, a small number of columns, between 10 000 and 15 000, with best reduced costs are chosen for the initial integer phase. Branching is done as described in the previous paragraph by locking follow-ons to segments. New columns are generated whenever the number of columns becomes too small or if the linear programming objective changes too much. In this way, we continue until at some point every segment has a unique follow-on. When this happens, we have found a feasible integer solution. For problems with a large number of segments, the linear program may become infeasible before every segment has been connected to a follow-on. We then generate more columns to resolve the infeasibility. Needless to say, the integer methodology is empirically derived. Other variants may work as well, but on the problems in our test set this procedure seemed to work best.

Maintaining integer feasibility in set-partitioning problems of this nature is quite difficult. In essence, our procedure can be said to convert it into a set-packing problem (with "less than or equal to" rather than "equal to" constraints). Covering may seem to be more natural in that it can be interpreted in this problem as deadheading (flying the crew as passengers). However, for the domestic daily problem, AA does not normally permit deadheads. In the later phases of planning, deadheads may be incorporated. Nevertheless, even if deadheads are allowed, they cannot be modeled exactly by simply changing the "equals to" to "greater than or equal to" because the pairing costs and legality are affected by a segment being flown as a deadhead. An exact formulation would keep the set-partitioning structure but allow segments in a pairing to be either deadheads or working segments. A deadhead segment in a pairing matrix then does not cover the segment since it must be flown by a working crew.

One final word about crew-base constraints. These constraints require that the total flying hours of all pairings originating and ending at a given crew base are within the range of available

flying hours of crews based there. Obviously, the linear program can incorporate these constraints, and since the pairings from the linear programming constitute the majority of pairings and the crew-base constraints are satisfied for them, any crew-base violations introduced can be easily repaired in the usual way as done by TRIP.

Example. Let us suppose that there are eight segments listed by departure station, departure time and arrival time, and arrival station:

1. DFW 900–1200 LGA
2. LGA 1300–1500 ORD
3. ORD 1600–1800 RDU
4. ORD 1700–1900 DFW
5. RDU 1900–2100 LGA
6. RDU 1900–2100 DFW
7. LGA 1400–1600 ORD
8. DFW 1600–1800 RDU

For this example, the times are rounded to hours, and all are considered to be in the same time zone. Consider the pairings:

1. DFW 900–1200 LGA 1400–1600 ORD 1700–1900 DFW
2. LGA 1300–1500 ORD 1600–1800 RDU 1900–2100 LGA
3. ORD 1600–1800 RDU 1900–2100 DFW 900–1200 LGA 1300–1500 ORD
4. DFW 1600–1800 RDU 1900–2100 DFW
5. DFW 1600–1800 RDU 1900–2100 LGA 1400–1600 ORD 1700–1900 DFW

There are two overnight rests: DFW in pairing 3, and LGA in pairing 5. The coefficient matrix for these eight segments (rows) and five pairings (columns) is

	1	2	3	4	5
1	1		1		
2		1	1		
3		1	1		
4	1				1
5		1			1
6			1	1	
7	1				1
8				1	1

A linear programming solution is $x_1 = x_2 = x_3 = x_4 = x_5 = 1/2$. The follow-on segments in this solution are given below with corresponding linear programming solution values in parentheses.

Segment	Follow-ons
1	7 (1/2), 2 (1/2)
2	3 (1)
3	5 (1/2), 6 (1/2)
4	1 (1/2), 8 (1/2)
5	2 (1/2), 7 (1/2)
6	1 (1/2), 8 (1/2)
7	4 (1)
8	6 (1/2), 5 (1/2)

In listing follow-on segments, the segment ending a pairing is considered to be followed by the segment beginning the pairing. Thus, the pairing is thought of as a circuit. Note that, in this example, even though all of the x_js are fractional, there are two segments, 2 and 7, with unique follow-ons in the pairings with positive x_js. Our procedure would begin by fixing these follow-ons so that any subsequent pairings generated will have (2,3) and (7,4) as locked pairs of segments.

Conclusions

We have demonstrated large improvements in crew pairing by initially generating millions of pairings and solving a linear program with a column for each pairing. This global approach seems to avoid local optima that prevent TRIP from making further improvements. For these large linear programs, we have consistently observed a large improvement in the linear programming objective value and the best TRIP-generated solution. Furthermore, the linear programming solution has many fractional variables. Finding good integer solutions is difficult, but the procedure presented here provides a practical and effective solution procedure.

Currently, the codes developed implementing this approach are being used at AA for the two largest fleets of planes: the super 80s and 727s. Starting from the best TRIP solution, improvements amounting to $300,000 have been realized for the months of August, September, and October. Pay and credit have been significantly reduced. This reduction has been particularly sig-

nificant in the summer months when traditionally crews are in short supply. The savings so far amount to 1.2 million dollars per year. When the same methodology is used to handle crew balancing, weekly exceptions, and transitions, and is applied to the smaller fleets as well, savings should be several times greater. In addition, we expect to improve on the optimization procedure. Thus, the approach has proven itself in practice and opens the possibility for moving toward global optimization of crew pairing.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. R. Anbil, E. Gelman, B. Patty, and R. Tanga, "Recent Advances in Crew-Pairing Optimization at American Airlines," *Interfaces* **21**, No. 1, 62–74 (1991).
2. G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization,* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Inc., New York (1988).
3. D. M. Ryan and J. C. Falkner, "A Bus Crew Scheduling System Using a Set Partitioning Model," *Asia Pacific Journal of Operational Research*, No. 4, 39–56 (1987).

Accepted for publication September 19, 1991.

Ranga Anbil *American Airlines Decision Technologies, Mail Drop 3F22, P.O. Box 619616, Dallas–Fort Worth Airport, Texas 75261.* Dr. Anbil is a principal at AADT managing the Crew Scheduling Research Group. Before joining AADT in 1986, he was an assistant professor at North Dakota State University in the Industrial Engineering Department. He received his Ph.D. from Ohio State University in industrial engineering in 1984. At AADT Dr. Anbil manages the various crew-pairing projects using TRIP.

Rajan Tanga *American Airlines Decision Technologies, Mail Drop 3F22, P.O. Box 619616, Dallas–Fort Worth Airport, Texas 75261.* Mr. Tanga is a senior consultant at AADT currently on leave to study at the Wharton School of the University of Pennsylvania. He joined AADT in 1988 after receiving his master's degree in industrial engineering from Ohio University. He received his bachelor's degree in mechanical engineering with highest honors from Gulbarga University in India. Mr. Tanga has worked mainly at applying mathematical technologies to complex scheduling problems.

Ellis L. Johnson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598-0218.* Dr. Johnson is an IBM Fellow in mathematical programming. He is also the Coca-Cola Professor of Industrial and Systems Engineering and Co-director of the Computational Optimization Center at Georgia Institute of Technology. Currently working in the area of mathematical programming, he established and managed the Optimization Center in the Mathematical Sciences Department at IBM Research from 1986 until 1990, during which time he oversaw the technical aspects of the development of the Optimization Subroutine Library (OSL). His work in integer programming has included projects with General Motors and American Airlines, and he contributed to the codes for MPSX.V2 MIP and OSL MIP. The paper describing the methodology developed to solve the General Motors problems received the Lanchester Prize from ORSA/TIMS in 1983. In 1985, Dr. Johnson was awarded the Dantzig Prize of the Mathematical Programming Society and the Society of Industrial and Applied Mathematics for his mathematical programming research, and in 1987 he was elected to the National Academy of Engineering.

Reprint Order No. G321-5462.