HIGH PERFORMANCE AIRLINE CREW-PAIRING OPTIMIZATION

CHRISTOS GOUMOPOULOS*
Dept. of Electrical & Computer Eng.,
Univ. of Patras, GR-26500, Greece
goumop@ee.upatras.gr

PANAYIOTIS ALEFRAGIS
Dept. of Electrical & Computer Eng.,
Univ. of Patras, GR-26500, Greece
alefrag@ee.upatras.gr

EFTHYMIOS HOUSOS
Dept. of Electrical & Computer Eng.,
Univ. of Patras, GR-26500, Greece
housos@ee.upatras.gr

ABSTRACT

Crew-pairing optimization involves the creation of a feasible and as close to optimum set of trips that cover all the flying activity of an airline. The problem is NP-complete which makes it computationally intractable and its solution requires specialized algorithms and heuristics. A major goal of the HPCN Esprit project PAROS is to improve the speed of the crew-pairing optimization process for the solution of very large problems. In this paper the ability to efficiently solve large crew pairing problems on a network of workstations (NOW) is presented. The main components of the crew-pairing optimization process are the pairing generator and the pairing optimizer. Large crew pairing problems from Lufthansa have been solved with a near linear speedup on the generator and satisfactory results on the optimizer.

KEYWORDS

parallel/distributed system, integer optimization, crew scheduling, heterogeneous network computing.

1. INTRODUCTION

Every airline must every day cover all of its planned flying activities by creating a work schedule for all of its crew members. In the HPCN Esprit project PAROS the user airline is Deutsche Lufthansa Airlines (LH), which employs more than 3000 pilots and 12000 flight attendants to fly one of the largest fleets in Europe and world wide with over 200 aircrafts and over 6200 flights per week. Total crew cost, which includes salaries, benefits, and expenses, is for every airline one of the largest operating costs. Therefore, a priority of the crew management department at LH, as well as in every other large airline, is to develop crew-scheduling processes that achieve a high degree of crew utilization. To meet this goal, crew planners have come to rely heavily on software systems, spending hundreds of computing hours each month for the solution of the problems. LH creates schedules for its crew members every month. Each flight in the month, also known as *leg*, has specific crew requirements. To make the problem even more complex, the crew members reside in ten different cities also called crew bases in the airline industry. The scheduling of crews to particular flights is done in sequences that start and end at the same base. These sequences of flights are called *pairings.*

Crew pairing optimization is an enormously complex combinatorial problem because the set of possible pairings is innumerable [1]. All large airlines find it impossible to efficiently solve this problem globally. A solution to the crew pairing optimization problem is a number of pairings such that all flights in the planning horizon are covered with the minimum cost. This is the fully dated problem since it specifies particular calendar days for the operation of all pairings. The fully dated problem is very large and it is solved by first solving a daily and a weekly approximation to the problem. The daily problem assumes that the flight schedule is the same every day. Once a daily solution is found it is interpreted on the weekly schedule and pairings that include flights that do not operate every day are re-optimized. In the same manner a weekly problem solution is rolled out to the fully dated problem by re-optimizing again the exceptions.

The crew pairing optimization system used at LH, is the Carmen automatic pairing construction system [2]. The Carmen system is also used by most of the major European airlines. The Carmen system is based on an approach in which the solution is iteratively improved by selecting and solving a series of subproblems [3]. However, the runtime required for the solution of large fleets is quite long and often the last available solution is used due to specific time constraints. For example, at present, a typical daily problem involving the scheduling of a medium size LH fleet requires about 15 hours of computing time, while for large problems, such as the monthly schedule, as much as 150 hours are needed. In contrast, LH would prefer to schedule their crews as fast as possible which in turn would allow them to start the scheduling process as late as possible, and thus minimize the negative effects of schedule changes.

*Corresponding author

The availability of fast workstations has allowed the airline companies to reduce the use of mainframes and thus significantly reduced their operational costs. However, the proliferation of workstations is in part responsible for the reduced efficiency of the hardware usage and the possible hardware infeasibility for large problems. This is in part resolved with the use of faster networks, and more efficient message passing systems, which allow for the full exploitation of idle CPU cycles and other global resources. These networks of workstations (NOW) can be used to execute in parallel most of the time consuming components of the crew pairing problem. This is the precise problem that the PAROS Esprit project is also attempting to address [4].

In the context of the PAROS (Parallel Large Scale Automatic Scheduling) project the consortium is faced with the challenge of improving the performance and extending the functionality of the Carmen crew scheduling process on a NOW. Lufthansa possesses a large number of powerful workstations connected with a fast network. The results from the first project prototypes demonstrate that it is possible to significantly improve the speed of the crew scheduling process and extend the practical range of solvable problems.

The remaining of the paper is organized as follows. In section 2 we present the solution methodology of the Carmen system. An analysis of the serial system is also given in order to reveal the possibilities for parallel computation. In section 3 we discuss the parallelization approaches adopted for the pairing generation and the pairing optimization components of the system. The results realized from the parallelization are given in section 4. Finally, conclusions and suggestions for future work are discussed in section 5.

2. CREW PAIRING SOLUTION METHODOLOGY

Because of the combinatorial explosion of the crew pairing problem, the present industry practice is to solve a series of subproblems and combine the intermediate results. The quality of the final solution depends on the number of iterations performed (limited by the available computer time) and also depends on the heuristics implemented for the subproblem selection. While there is no way of measuring how close the final schedules obtained are to optimality, this method gives practical and useful results.

2.1 CARMEN APC SYSTEM

The automatic pairing construction of the Carmen system is called APC and each of its iterations involves three basic steps: *subproblem selection*, *pairing generation*, and *pairing optimization* (Figure 1). In the subproblem selection step, heuristics that are mostly based on manual crew planning methodologies and problem characteristics contribute to the reduction of the number of generated pairings and to the minimization of the local solutions impact [3].

In the pairing generator, basically consisting of a depth-first search algorithm, the search tree, determined by the connection matrix, is pruned based on the user controlling parameters and a complex set of rules in order to avoid excessive generation of useless or illegal pairings. Each pairing is associated with a cost value. LH uses a complicated set of rules to determine the cost of a pairing. Costing is based on attributes such as the number of duty periods, the total time spanned by the pairing, and the number of *deadhead* hours (hours that the crews fly as passengers).

The third step is the selection of the best solution pairings. Given a set of legal pairings, generated in the previous step, one can formulate the problem of finding the best collection of pairings, such that each flight is covered by at least one pairing, as a set covering problem [5], where the variables correspond to pairings and the constraints correspond to the actual legs to be covered.

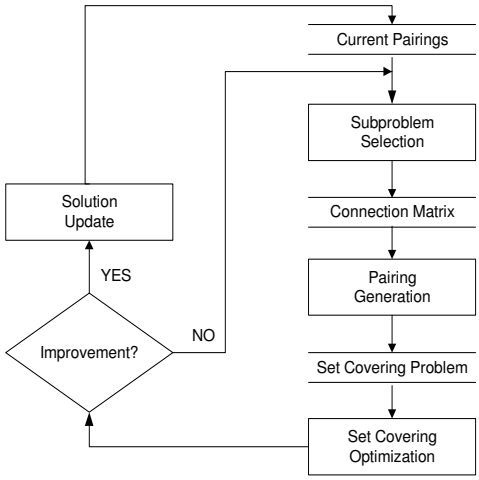

FIGURE 1. THE CARMEN APC SUBPROBLEM METHODOLOGY

The number of legs covered in every iteration is from 300 up to 1,200 for the large daily problems. As many as 1,000,000 pairings are often generated. In order to handle the very large problems created by the generator, a set covering optimizer which works well for very large problems is used [6]. A typical run of the Carmen APC system consists of 50-100 iterations. Within each iteration and for all of the Lufthansa fleets that were tested, the generator takes 5-8 times more than the optimizer. This is partly due to the large and complex number of legality checks the generator must perform and partly due to the speed and performance of the optimization algorithm. The memory requirements of the Carmen system are between 128 - 512MB for relatively large problems.

2.2 SERIAL SYSTEM ANALYSIS

In the Carmen APC system the generator and the optimizer could both run faster if parallel processing techniques were used. Profiling analysis of the serial system reveals that the pairing generation phase takes, depending on the type and the size of the problem, 70-85% of the runtime required for the solution of each problem. Since there exists a pairing generation algorithm, which is fully amenable to parallel computation, we expect significant runtime reductions of the generation phase when a large number of networked workstations is used. The set covering optimization step takes 10-20% of the total runtime and it appears to be less amenable to efficient parallelization. A parallelization strategy for this phase will be discussed later. However, the optimizer can exploit the global memory resources of the NOW which is extremely important for large dated problems. Finally, 5-10% of the time is consumed in the analysis and the subproblem selection that is inherently a sequential process. This gives an upper bound to the speedup of the overall solution process equal to 10-20 irrespectively of the number of processors used.

3. PARALLEL CREW PAIRING OPTIMIZATION

As a result of the previous analysis, the parallel crew pairing optimization system will consist of three components: the parallel generator, the parallel optimizer and the sequential subproblem selection process.

A major goal of the PAROS project, as stated before, is to improve run times for solving large problems. To be successful in improving speed by parallelization and developing new heuristics, it is of great importance to have a data model that gives fast access methods for the most frequently used operations, but is also memory efficient. Therefore the data model of the system has been changed to be also more memory efficient. In the remaining of this section we will describe the parallelization strategies for both the pairing generation and pairing optimization steps.

3.1 PAIRING GENERATION COMPONENT

Figure 2 shows an example of a typical pairing that starts and ends in Frankfurt (FRA), an LH crew base, displaying such pairing attributes as duty periods, overnight rests, and briefing and debriefing times. The construction of crew pairings is complicated by a complex set of union, company and governmental rules and regulations. These rules vary by crew type (pilot or flight attendant), crew size, aircraft type, and type of operation (domestic or international). Work rules concern duty periods and rests. A stringent union rule specifies maximum duty length, which varies between 14 and 16 hours. Other duty rules govern the maximum flying time allowed and the maximum number of flights permitted. The governmental regulations minimize crew fatigue and ensure passenger safety. Minimum rest requirements are tied to the flying time scheduled in a moving 24-hour window.

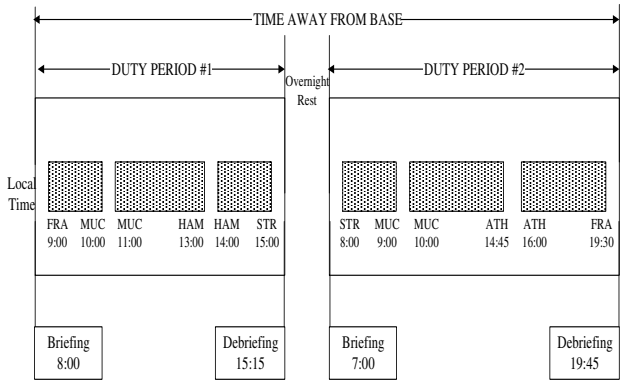

FIGURE 2. A TYPICAL PAIRING

3.1.1 THE PAIRING GENERATION ALGORITHM

The generator creates a large number of pairings by connecting legs to each other in different combinations. One of the most basic and conceptually clear algorithm for the generation process is a depth first search in a search tree stored in a data structure known as the connection matrix. The connection matrix represents in mathematical terms a directed connection graph among the legs. A node of the graph corresponds to a leg and an edge represents a legal pair-wise connection. The possible connections for every leg are organized in a priority list based on the special characteristics of each connection. The search always begins from a subset of legs known as *start legs*, being those legs that depart from a crew base. In a similar manner *end legs* leading back to the same crew base can be identified for every start leg. The concurrency is here since the generation for each start leg is independent of the generation of every other start leg. The size and the shape of each tree that must be explored are not known ahead of time. This makes load balancing extremely important.

The generator search is limited by a maximum number of branches considered in each node of the search tree. This maximum number of connecting branches is known as the *search width* and its typical value range from 5-8 connections (Figure 3). In every step of the depth first search the created sequence is checked for legality. Illegal paths are not investigated further assuming that the rules have a monotonic behavior. This implies that if a sequence is illegal it can not become legal by adding more legs to it. Depending on the rules, and the size and structure of the problem, the generation time in each iteration could take from a few minutes to many hours.

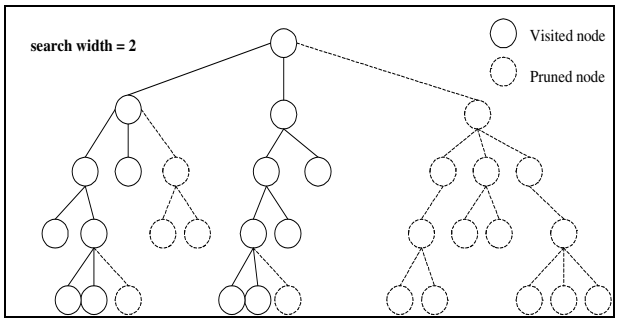

FIGURE 3. SEARCH TREE AND SEARCH WIDTH

3.1.2 PARALLELIZATION STRATEGY

The parallelization of the pairing generator is based on a manager/worker scheme. The manager distributes dynamically the start legs and the necessary additional problem information to the workers on a demand driven manner. The workers must generate all the legal pairings and return them to the manager. The goal is to minimize the idle time and the communication. To minimize idle time we do load balancing and we do overlapping between computation and communication. To minimize communication we use large messages (buffering of pairings in the workers) in order to reduce the high network latency penalty.

Load balancing is achieved by implementing a dynamic workload distribution scheme that implicitly takes into account the speed and the current load of each machine. The number of start legs that are sent to each worker is also changing dynamically with a fading algorithm. In the beginning a large number of start legs is given and in the end only one start leg per worker is assigned. Efficiency is also improved by pre-fetching. A worker is requesting the next bunch of start legs before it is needed. It can then perform computation while the manager is servicing its request. Fault tolerance has also been implemented. Parallel generator can recover from task and host failures. For example, a worker failure is detected by the manager which also keeps the current computing state of the worker. In case of a failure the state is used for re-assigning the unfinished part of the work to another worker.

We have developed a performance model in order to evaluate the parallel generator algorithm. We measure primarily the efficiency of the parallel generator. From this metric the speedup can also be obtained. The model uses a number of assumptions for the shake of simplicity and in-order to abstract insignificant details. We assume: idle time very close to zero, all processors equal and not loaded, unloaded network, no competition for bandwidth among the workers (true for asynchronous algorithms), insignificant communication cost of generation start-up.

We can easily prove that the efficiency and the speedup of the parallel generator is a function of the granularity, G of the problem.

$$E_{Generator} = \frac{G}{1+G} \tag{1}$$

$$S_{Generator} = P \cdot \frac{G}{1+G} \tag{2}$$

where P in (2) is the number of processors used. The granularity is defined as the computation time (T_{comp}) to communication time (T_{comm}) ratio. It is given also from the following equation:

$$G = \frac{EfBa}{gdr + \frac{P}{T_{comp}} \cdot CMsize} \tag{3}$$

The granularity of the problem depends mainly on two parameters, the effective bandwidth (*EfBa*) and the generation data rate (*gdr*). The second term in the denominator of (3) represents the cost to broadcast the connection matrix (*CMsize*) to the workers in the start-up phase and usually it is not important.

The generation data rate depends on the complexity of the rules and the speed of the processor. Profiling analysis shows that for typical problem parameters the generator produces about 80 pairings per second on a high end workstation. This gives a data rate of about 8 KB/sec since each pairing requires 100 bytes to encode. This figure compared with the 1 MB/sec bandwidth of standard Ethernet gives a high granularity value. Consequently, parallel generation is fully scalable and we expect high efficiency, and linear speedup for a large number of workstations even on an Ethernet network. Of course, contention from other users in the network will influence the parallel generation performance.

3.2 PAIRING OPTIMIZATION COMPONENT

The optimization algorithm is an approximation solver for 0-1 ILP (Integer Linear Programming) problems. At present, it is used within the Carmen APC system to solve large scale set covering problems with base capacity constraints [2], that are created by the pairing generator. This problem can be expressed mathematically as follows:

$$\begin{aligned} &\min cx \\ &Ax \geq 1 \quad \text{(set covering constraints)} \\ &Dx \leq d \quad \text{(base capacity constraints)} \\ &x = 0, 1 \end{aligned}$$

where A is a 0-1 matrix and D is arbitrary non-negative. This problem is NP-hard, which indicates that finding the optimal solution may be extremely time consuming for large problems.

The mathematical foundations of the serial algorithm are described in [6]. In brief, the algorithm works as a simple dual ascent method, designed to use only operations that are efficient for large problems. An approximation scheme that manipulates the cost vector c as little as possible to give 0-1 solution to the primal problem is used. The magnitude of the manipulations is controlled by a parameter κ, which is slowly increased from 0 up to its maximum value of 1 during each iteration. A high level description of the algorithm follows:

Optimization algorithm

$\bar{c} = c$, reset all s^i to 0, $\kappa = 0$
repeat
  for every constraint i
    $r^i = \bar{c}^i - s^i$
    s^i = a function of r^i, b_i and κ
    $\bar{c}^i = r^i + s^i$
  increase κ according to a heuristic rule
until no sign changes in $\bar{c}$

where $\bar{c}$ are the reduced costs that the algorithm operates on, $\bar{c}^i$ is the sub-vector of $\bar{c}$ corresponding to the non-zero elements of constraint i, s^i is the contribution to $\bar{c}$ by constraint i.

3.2.1 PARALLELIZATION STRATEGY

Since the nature of the algorithm is iterative, parallelization has to work within the iterations. The iteration over the constraints is selected for parallelization. The idea is to let every processor have a local copy of the reduced cost vector and be responsible for a subset of the constraints in the original problem. This will also distribute the memory requirements, which are important for large problems. To ensure similarity between the local copies communication is required.

Mathematically, we duplicate the variables x to x, x', x'', … so that each processor receives its own copy of variables, and add the constraints $x = x' = x'' = \ldots$ to the problem. We also partition the original constraints (A, b) into groups (A', b'), (A'', b''), etc.

The efficiency of the above scheme depends on the relation between the computation time and the communication time (granularity). With strict synchronization and full updating of the constraints $x = x' = x'' = \ldots$ after every iteration of a single constraint, it is possible to emulate the exact behavior of the serial algorithm. On the other hand, very little communication would lead to convergence problems and poor solution quality. However, under certain circumstances, a compromise could give both good solutions and significant speedup.

In principle, two constraints that do not share any variables can be iterated independently. In this effect, a partitioning strategy that allocates constraints to the given number of processors in such a way that the amount of communication is minimized is desirable. However, this must be done under the restriction that constraints should be evenly balanced on the processors. For set covering problems with a block diagonal structure, the need of updating through communication will be significantly lower than for random problems, provided that the blocks can be successfully assigned to the given number of processors. In practice, some problems that arise from merged fleets could have such structure to a certain degree.

The problem of optimal partitions is related to the well-known NP-hard problem of graph partitioning [7]. However, we do not essentially need an optimal partitioning solution, but only a reasonable approximation. Several software packages for this problem exist like METIS, PARTY, JOSTLE, Scotch. We use a fast greedy algorithm with node exchange. A specific observation for the fully dated problems of LH is that sorting the pairings by departure time of the first leg should result to a natural partition.

A significant observation that can justify further relaxation of the communication is the following. Dependencies between constraints are only important if they involve critical variables [8]. If a group of constraints do not have any common critical variables, and the iteration of each constraint, does not change the critical variables of any other constraint, the constraint calculations can be made independently of each other, and will give the same result as serial computation in any order. Based on this, a very relaxed protocol for updating the reduced cost vector sparingly in a more coarse-grained fashion was tested with mixed results. Experiments have shown that the magnitude of relaxation has to be carefully tuned in order to achieve quality solutions. A high magnitude can give speedups but low quality. A lower one can guarantee convergence and quality but gives slow execution. An update of the cost vector after the iteration of 5-15 constraints, depending on the size and the structure of the problem, has empirically been found acceptable.

The most important result of this parallel algorithm is that it is able to find solutions of similar quality compared to the serial version, in spite of the fact that updates of constraints on different processors are done independently of each other. This also indicates that the strategy will be able to solve larger problems than the ones we can solve on a single machine, exploiting the distributed memory, which will by itself can be useful even without a speedup. For instances of problems where a block structure can be found a maximum speedup of 2 has been achieved on 4 processors. However, only a few problems can have this

property, and for the majority of the problems only a limited number of independent constraints can be found.

4. IMPLEMENTATION FRAMEWORK AND EXPERIMENTAL RESULTS

For the implementation of the parallel generator and the parallel optimizer the PVM message passing system was used [9]. PVM (Parallel Virtual Machine) is a software framework that emulates a generalized distributed memory multiprocessor in heterogeneous networked environments. The system is portable and provides support for heterogeneity, robustness, and a simple but complete API. In addition auxiliary systems, including monitoring and debugging tools [10] (XPVM, TotalView, etc.), are available and have been used for the development of the parallel system. The notification mechanism of PVM was found particularly useful since it enabled application level fault tolerance to be incorporated in the implementation.

The experiments have been performed on a heterogeneous network of HP workstations (715/50, 715/100, 735, 750 and C-180), connected by Standard Ethernet at the computing center of the University of Patras. The execution times presented were obtained during nights when almost exclusive usage of the network and the workstations was possible. We have measured the speedup of the parallel generator and optimizer using as reference processor (RP) the HP 715/100 model, see the discussion about speedup and efficiency for a heterogeneous network.

For our experiments we used three typical LH problems. The characteristics of these problems are given in Table 1. The size of an input problem for the parallel generator is determined by the number of start legs, while the size of an input problem for the parallel optimizer is determined by the number of the generated pairings.

Problem Name	No of start legs	Set-up overhead	Generated pairings
LH1	1366	1.2 sec	43932
LH2	2984	2.5 sec	97075
LH3	5968	4 sec	194150

TABLE 1. TYPICAL PAIRING PROBLEMS

In Table 2 and Table 3 we report the elapsed time and the speedup achieved for the parallel generator and the parallel optimizer respectively. In Table 2, #(Net, RP) denotes the relative power of our heterogeneous network with respect to the RP processor. The results have been slightly approximated for the shake of the presentation. For the parallel optimizer it was possible to use four identical workstations.

Speedup and Efficiency for a Heterogeneous Network

The speedup (1) and efficiency (2) on homogeneous networks is naturally relative to any of the similar processors.

$$S(N) = \frac{T(1)}{T(N)} \quad (1) \qquad E(N) = \frac{S(N)}{N} \quad (2)$$

where N is the number of processors and $T(1/N)$ is the execution time of the parallel algorithm on 1/N processors.

For heterogeneous networks of processors, the speedup can no more be implicitly relative to the execution time on one processor but must be explicitly relative to the execution time on a *reference processor* (*RP*), because all processors are no more identical. Let RP be the processor used for the execution of the sequential algorithm. We define the speedup of a parallel algorithm for the heterogeneous case as:

$$S(Net, RP) = \frac{T(RP)}{T(Net)} \quad (3)$$

where *Net* denotes the network, $T(RP)$ is the execution time of the sequential algorithm, and $T(Net)$ is the execution time of the parallel algorithm on the network. Then we define the relative power #(Net, RP) of a heterogeneous network with respect to processor RP as

$$\#(\text{Net, RP}) = \sum_{i=1}^{|Net|} \gamma_i \quad (4)$$

where γ_i is the normalized processor speed defined as the ratio between the execution time of the algorithm on processor i and the execution time of the algorithm on processor RP, and |Net| is the number of processors in the network. Then #(Net, RP) can be seen as the equivalent number of reference processors for the algorithm that is considered. We can prove that S(Net, RP) $\leq$ #(Net, RP).

The efficiency of the parallel algorithm will be:

$$E(Net) = \frac{S(Net, RP)}{\#(Net, RP)} \quad (5)$$

#(Net,RP)		1	2	4	5	7	8	10
LH 1	*T(Net)*	34.5	18.2	8.8	7.3	5.2	4.5	3.6
	S(Net,RP)	1	1.89	3.9	4.7	6.65	7.6	9.7
LH 2	*T(Net)*	76.6	39.8	20.2	16.1	11.4	10.2	8.0
	S(Net,RP)	1	1.92	3.8	4.75	6.72	7.52	9.6
LH 3	*T(Net)*	153	79.7	39.2	32.0	22.6	20.2	16
	S(Net,RP)	1	1.92	3.91	4.8	6.79	7.6	9.6

TABLE 2. ELAPSED TIME *T (Net)* (min) AND SPEEDUP *S(Net,RP)* FOR A TYPICAL ITERATION OF THE PARALLEL GENERATOR

	N	1	2	3	4
LH1	*T(N)*	123	117	145	167
	S(N)	1	1,05	0,85	0,74
LH2	*T(N)*	256	147	156	174
	S(N)	1	1,74	1,64	1,47
LH3	*T(N)*	314	254	289	293
	S(N)	1	1,24	1,09	1,07

TABLE 3. ELAPSED TIME *T(N)* (sec) AND SPEEDUP *S(N)* FOR A TYPICAL RUN OF THE PARALLEL OPTIMIZER

For the generator the speedup in all cases is almost linear to the number of processors, while the optimizer can only achieve slight speedups. Given that the Carmen system generator, which is currently in production at LH, requires about 80% of the total system runtime, the overall performance will be significantly improved. For example, a crew scheduling problem that requires 20 hours to solve completely, would take about 5.5 hours with 10 identical machines and 4.8 hours with 20 machines assuming that everything else remained as in the current Carmen system.

5. CONCLUSIONS AND FUTURE WORK

Crew pairing optimization is one of the major problems that an airline company must solve. It is time consuming and deadline driven. In this paper we have presented strategies and specific results that improve and extend the successful Carmen APC system with the use of parallel processing on a NOW. The pairing generation phase is the dominant part of the system and can be efficiently parallelized on a large number of workstations, achieving a significant performance improvement. A nearly linear reduction of the pairing generation runtime is possible due to the highly parallelizable pairing generation procedure. The set covering optimization phase can benefit from the global memory resources of the NOW for large problems. For problems that exhibit a block like structure moderate speedup can also be achieved on a small NOW.

The improvements in performance translate immeiatey in significant financial and strategic benefits for the airlines. The ability to start the planning process as close to the day of operation as possible allows for fast market reactions and reduces the need for continuous rescheduling, thus reducing the solution quality, due to unexpected aircraft assignments and other schedule changes. Faster solution times translate to higher productivity of the airline crew management department since crew schedules can be created closer to the actual day of operation, which is very important in the new deregulated environment in Europe and elsewhere. In addition, the high performance of the parallel generator allows larger problems to be solved since in the sequential system the slow runtime was the constraining factor.

Future work includes an investigation of an alternative parallelization strategy for the pairing optimization phase that distributes the variables and not the constraints over the processors. This approach has the advantage that small volume of data must be communicated between the processors. However, the need to exchange several small messages, makes the approach extremely sensitive to the network communication latency. Finally, a better and more direct integration between the parallel generator and the parallel optimizer is being investigated.

REFERENCES

[1] G.W. Graves, R.D. McBride and I. Gershkoff, Flight Crew Scheduling, *Management Science*, 39(6), 1993, 736-745.

[2] E. Anderson, E. Housos, N. Kohl and D. Wedelin, *OR in the airline industry*, Crew Pairing Optimization chapter, 228-258, Boston, London, Dordrecht, Kluwer Academic Publishers, 1997.

[3] E. Housos and T. Elmroth, Automatic Subproblem Optimization for Airline Crew Scheduling, *Interfaces* 27:5, 1997, 68-77.

[4] PAROS consortium, *Technical annex for the Esprit project 20.115: Parallel large scale automatic scheduling*, PAROS, January 1996.

[5] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley - Interscience, 1988.

[6] D. Wedelin, An algorithm for large scale 0-1 integer programming with application to airline crew scheduling, *Annals of Operations Research*, vol. 57, 1995, 283-301.

[7] A. Gupta, Fast and effective algorithms for graph partitioning and sparse-matrix ordering, *IBM Journal of Research & Development*, 41(1/2), 1997, 171-184.

[8] PAROS consortium, *D5.3 Parallel optimizer prototype*, University of Patras, May 1997.

[9] G. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck and V. Sunderam. *PVM: Parallel Virtual Machine*, MIT Press, 1994.

[10] A. Beguelin and V. Sunderam, Tools for Monitoring, Debugging, and Programming in PVM, *Proceedings of EuroPVM 96,* Munich, Germany, Oct. 1996, 7-13.