Parallel Algorithms for Airline Crew Planning on Networks of Workstations*

Christos Goumopoulos, Panayiotis Alefragis, Efthymios Housos
Department of Electrical & Computer Engineering, Computer Laboratory
University of Patras, GR-265 00 Rio Patras, Greece
{goumop, alefrag, housos}@ee.upatras.gr

Abstract

The crew planning problem has been successfully solved on a loosely connected network of workstations (NOW) using advanced computational techniques and efficient communication patterns. The parallelization of the successful sequential system of Carmen Systems AB guarantees that the results are immediately useful and applicable to a large number of airlines scheduling problems. The parallel pairing generator component of the crew scheduling process achieves a linear speedup on the number of processors and can be efficiently scaled to a large number of processors. The novel parallel optimizer approach of the paper also achieves almost linear speedups for large problems solved on a small number of workstations. The Lufthansa problems that were used in our experiments validate our theoretical results and prove the value and usefulness of our work.

1. Introduction

The use of resource planning optimization techniques in industrial applications is imperative for the present competitive environment of the global economies and can significantly reduce operational costs. A typical example is the transportation industry where scheduling applications like crew scheduling [9] and crew rostering [4] lead to very large and difficult optimization problems with long computation times. Solving such problems in acceptable time with exact algorithms is impossible due to the combinatorial explosion that characterizes most of these problems. Heuristics are often used in order to reduce the search space and improve the computational tractability of these problems. In any case the run times of these procedures are very high, which makes the use of parallel processing imperative [3]. In addition the use of parallel processing is driven by the fact that close to day of operations solutions are also desired due to the continuously changing business environment.

The application example and the algorithms described in this paper were in part supported by the European ESPRIT/HPCN project PAROS (Parallel Large Scale Automatic Crew Scheduling) [1,24]. The project started in 1996 with Lufthansa Deutsche Airlines (LH) as the industrial user, Carmen Systems AB, the University of Patras and the Chalmers University of Technology as the other partners. LH supplied important large problems and optimization requirements in the area of crew planning. PAROS is an effort to improve the automatic crew scheduling system produced and marketed by Carmen Systems with the use of high performance computing and modeling techniques. Performance improvements will allow considering more realistic scheduling periods while giving the marketing department additional time to satisfy the market needs.

The network of workstations that has been selected as the parallel processing platform is a cost effective and widely available computation model. Computers connected through high performance networks can be used as parallel machines. The availability of faster workstations in the past ten years has allowed the airlines to reduce the use of mainframes and thus reduce their computational costs. This, however, did create a thrashing computational effect when it comes to large combinatorial problems. The use of networks of workstations for the solution of a single problem minimizes this effect. The emphasis was to develop new software for efficient coordination and cooperation of networked workstations to achieve higher productivity and faster solutions for the crew planning problem. While there exist attempts to solve the crew planning problem on high end parallel hardware [17], this paper focuses on issues that arise in parallelizing this problem on a cluster of workstations.

* This work has been supported by the ESPRIT HPCN program.

The rest of the paper is organized as follows. In section 2 we describe the crew planning problem and the prevailing solution methodology. The parallel algorithms developed for the two most time critical components of the solution process, the pairing generator and the pairing optimizer, are discussed in sections 3 and 4. Theoretical performance analysis of the proposed parallel algorithms is also presented. In section 5 experimental results from typical Lufthansa crew scheduling problems are shown. Finally, conclusions and future directions of this work are discussed in section 6.

2. Airline Crew Planning Process

The crew planning department receives the schedule from the aircraft scheduling department at regular time intervals and has to create legal round trips in such a way as to satisfy the crew requirements of all flights. The optimal set of trips must comply with the general safety regulations, the company operating policies and the union requirements, while minimizing the total cost. The basic activity to be planned is called *leg* and involves a single departure and a single arrival. Legs are connected into round-trips also called *pairings* that depart and return to specific crew bases. Given the flight table and the distribution of the crews among the crew bases, the planning process may be separated as follows:

- selection of the flight legs to be covered
- pairing generation and optimization
- assignment of the pairings to individual crew members

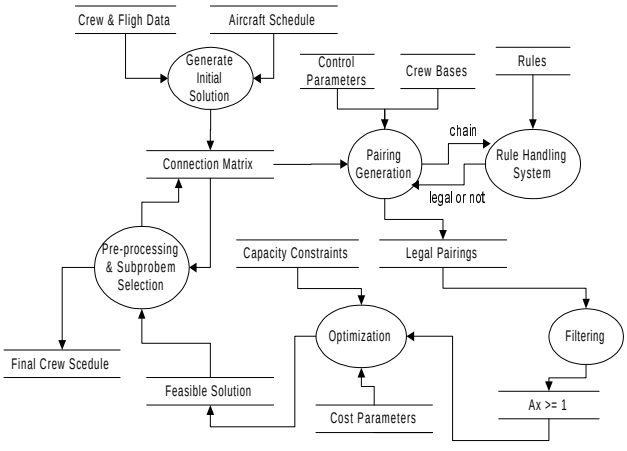

Figure 1. Carmen System DFD

Since the number of possible pairings extends into millions and possibly billions obtaining a meaningful set of pairings, to be given to the optimization procedure, is quite complex and time consuming. The generation of a meaningful set of pairings is aided either by special pre-processing filtering techniques [14] or optimizer feedback based processes [16]. The optimization phase involves the selection of a set of pairings in order to cover all flight legs while minimizing the total solution cost.

The solution methodology of the Carmen system [2] is shown in (Figure 1) and is presently in production at Air France, Alitalia, British Airways, KLM, Lufthansa, SAS, and Swissair. A typical run of the Carmen system consists of 50-100 iterations. The main system components are the pairing generation and the optimization modules.

The total number of possible pairings depends on the structure of the flight network. A typical short haul fleet for Lufthansa has about 800 daily legs and about 5000 weekly legs. A typical pairing in this fleet contains on the average twelve legs, and since a leg can be connected to at least ten new legs at each major airport, this produces close to 10^{12} pairings. To reduce the number of pairings requires an intelligent generation procedure. The optimizer that will be examined and parallelized in this paper is quite fast and achieves high quality results for problems up to one million pairings. Within the Carmen system and for all of the Lufthansa fleets that were tested, the generator always requires five to eight times more execution time than the optimizer or about 70-85% of the total runtime.

3. Parallel Pairing Generator

3.1. Pairing Generation Algorithm Description

The pairing generator creates legal pairings by connecting legs to each other. The pairing generator is aided by a pre-processed *connection matrix* that shows the acceptable connections between pairs of legs. In addition, there exists a legality module that calculates the properties of each chain and validates all the applicable rules. 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 non-zero elements of the matrix for a typical fleet of 1000 legs can be from 10,000 to 100,000.

The most efficient algorithm, with respect to memory needs, for the pairing generation process is the depth first search (Figure 2). The search always begins from a subset of legs known as *start legs*. The search is limited by a maximum number of branches to be considered in each node of the search graph. This maximum number of connecting branches is known as the *search width* and its typical value range from 10-20 connections. Every leg chain is checked for legality. Illegal paths are not further investigated which implies that the incremental rules must have a monotonic behavior.

Input: A set of start legs $S = \{s_1, s_2, \ldots, s_k\}$, Connection Matrix *CM*, set of rules *R*, search width *SW*(*P*) as a function of the working days covered by *P*, where *P* denotes a chain of legs

```
procedure GENERATE
    Work queue WQ ← S
    while (WQ not empty) do
        node ← GET_NEXT(WQ)
        SEARCH(node)
    endwhile
endprocedure
procedure SEARCH(node)
    P ← ADD(node)
    if TEST_LEGALITY(P, R) then
        if P is a COMPLETE pairing then
            OUTPUT(P)
        endif
        while SW(P) is not violated do
            r ← GET_NEXT_CONNECTION(node, CM)
            SEARCH(r)
        endwhile
    endif
    P ← REMOVE(node)
endprocedure
```

Figure 2. Serial Pairing Generation Algorithm

3.2. Parallelization Approach

An examination of the serial algorithm in Figure 2 reveals the ability for exploiting parallelism during the pairing generation phase. The majority of the computation time occurs in the *SEARCH* procedure. By distributing the contents of the work queue *WQ*, thereby dividing the computational work among several processors we can reduce the computation time. The amount of computational work done in the *SEARCH* procedure for each element of the *WQ* is highly variable and unpredictable. This implies that the parallelization must incorporate dynamic load balancing mechanisms.

The parallel programming approach used for the parallel generator is the manager/worker model. The manager executes the *GENERATE* procedure and the workers execute the *SEARCH* procedure. The manager broadcasts the connection matrix to every worker at the beginning of the run. This implies that the parallel process involves the distribution of a forest of search trees to the available workers. The manager distributes dynamically the start legs and the search width information to the workers on a worker demand driven manner. The workers generate all the legal pairings and return them to the manager. The communication between the manager and the workers is asynchronous and there is no need for communication between the workers. The manager is composed of two threads, one responsible for the distribution of the input data to the workers and the other for collecting the output from the workers. This scheme improves the efficiency and the scalability of the parallel generator, despite of the centralized nature of the manager. The typical mapping involves the assignment of each worker to a different processor.

The design goal of all parallel processing applications is to minimize the idle time of each worker and the communication among the processors. To minimize idle time application specific load balancing is done and an overlapping between computation and communication is attempted. To minimize communication we use large messages, that is, the workers do buffering and compression of the pairings in order to reduce the network latency penalty and the volume of the communicated data.

3.3. Application Specific Features

Dynamic Load Balancing. Load balancing is achieved by implementing a dynamic workload distribution scheme in the manager 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 are also changing dynamically using a fading algorithm. In the beginning a sufficient number of start legs is given and near the end only a single start leg is assigned to each worker. This scheme attempts to balance the network traffic and the load balancing sensitivities. In (1) the number of start legs (*n*) assigned to each worker as a function of the number of the remaining start legs (*r*) is shown.

$$n = \begin{cases} \max\left(1,\ \min\left(UB, floor(N_{TOTAL}/(P \cdot \log 2(N_{TOTAL}/P)))\right)\right) & if\ r = N_{TOTAL} \\ \max(1,\ f(r) \cdot n) & if\ 3nP \le r < N_{TOTAL} \\ 1 & if\ r < 3nP \end{cases} \quad (1)$$

N_{TOTAL} is the total number of start legs for the current iteration of the problem, *UB* is an upper bound for the initial work distribution and *f(r)* is a monotonic decay function in [0,1]. The initial work assignment depends on the number of processors (*P*) and N_{TOTAL} and is done simultaneously for all the workers. In addition, efficiency is also improved by pre-fetching the start legs from the manager. A worker requests the next set of start legs before they are needed. It can then perform computation while its request is being serviced by the manager.

Because the search tree that corresponds to each start leg may be very irregular a further refinement of the load balancing scheme is also implemented as the end of the pairing generation is approached. The manager decreases the granularity of the search tree at a lower level and assigns sub-trees to the workers (Figure 3).

Fault Tolerance. For production level reliability the parallel generator is able to recover from task and host

failures. The notification mechanism of PVM [11] is used to provide application level fault tolerance to the generator. A worker failure leads to the loss of some pairings that either have not been generated, or have been generated but not sent. Consequently, this part of the generation tree must be recalculated by some worker task. The manager keeps the current computing state of each worker and in case failure it is used for reassigning the unfinished part of the work.

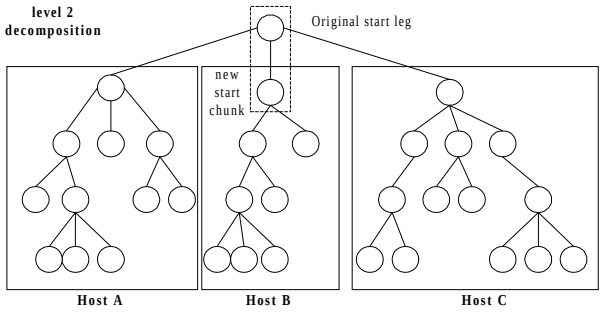

Figure 3. Decomposition of the search tree

The program that called the parallel generator detects the failure of the manager and can start a new manager, as the manager uses a checkpointing mechanism to store state information. The responsibility of the new manager is to reset the workers and to request only the generation of the pairings that have not been generated.

Dynamic Task Creation and Local User Priority. Another powerful feature of the parallel pairing generator is its ability to utilize the availability of new processor nodes and to return to their owners machines that must not be used in the parallel virtual machine any longer. It is possible to add a new host at any time to the virtual parallel machine, and this will cause a new worker to be started automatically. The system also respects the workstation owner's priorities. This is implemented with the suspend/resume set of services. In simple terms, the worker suspends its operation if the machine load (CPU, memory, swap space) is over some specified limits and resumes its operation if the machine load is below a specified limit. When a worker is in suspend mode it is considered blocked and the manager keeps this worker in a list of suspended tasks. Periodically, the manager requests from the worker information concerning its machine load and if the load is below the specified limit the manager moves the worker to the active list of workers. If the worker remains in suspend mode for a long period of time, it is considered as failing, and is removed completely from the system. The rate of the load checking operation defines the performance overhead; interaction of about once every minute creates a very small performance overhead (<0.1%).

4. Parallel Pairing Optimizer

4.1. Pairing Optimization Problem Definition

The pairing optimization problem is modeled as a *set covering* or *set partitioning* 0-1 constraint satisfaction problem [20]. A small number of general capacity constraints also exist but are not considered in this paper.

Let A = (a_{ij}) be a 0-1 m *x* n matrix and C = (c_j) be a vector of size n. Let the sets M = {1, … ,m} and N = {1, … ,n} correspond to the rows and columns of A. The value c_j ($j \in N$) represents the cost of column j (pairing j). We assume, without loss of generality, that $c_j \geq 0$, $\forall\ j \in N$. We say that a column $j \in N$ covers row $i \in M$, if $a_{ij} = 1$. We say that a subset of columns $S \subseteq N$ is a solution to the set covering problem, if for each row $i \in M$, there is at least one column $j \in S$ with $a_{ij} \neq 0$. The target of the optimization algorithm is to select the subset S that minimizes the sum of the corresponding c_j's. In mathematical terms the problem can be written as

$$\min \sum_{j \in N} c_j x_j \tag{2}$$

$$s.t. \sum_{j \in N} a_{ij} x_j \geq 1,\ x_j \in \{0,1\},\ i \in M,\ j \in N \tag{3}$$

where $x_j = 1$ if $j \in S$ and $x_j = 0$ otherwise. The large problems we were concerned have up to one million variables and typically between a few hundred to a few thousand constraints. They are very sparse, usually having only 5 to 10 nonzero entries per column.

The set covering problem is *NP*-hard, and many algorithms have been proposed for exact [10] as well as approximate problem solutions [7,27]. The exact approaches are often based on the branch and bound search technique, which although can be successfully parallelized on the NOW architecture [5,19], has a prohibitive computation time for very large problems. Since our goal was to solve large crew scheduling problems the approximate solution algorithm proposed in [27] was selected for parallelization. This is also the algorithm used in the current serial Carmen system and is particularly efficient for this class of problems.

4.2. High Level Optimization Description

The optimization algorithm can be categorized as an iterative Lagrangian relaxation heuristic algorithm [25]. The algorithm manipulates the Lagrangian *cost vector* $\overline{c}$, which is initialized to the cost vector C. The goal of the manipulation is to create a sign pattern for the elements of $\overline{c}$, which corresponds to a feasible solution S (usually well within 1% of the optimum), given that $\forall\ x_j \in S$, $\overline{c}_j < 0$. In the sequential algorithm all the problem constraints

are iterated, one at a time, updating the corresponding $\bar{c}$ entries. The algorithm is summarized in Figure 4.

Input: 0-1 mxn constraint matrix *A*, cost vector *C*

$\bar{c} \leftarrow C$, $\kappa \leftarrow 0$, $\forall\, i \in M\ s^i \leftarrow 0$
repeat
  for each constraint *i* **do**
    $r^i \leftarrow CANCEL(\bar{c}, s^i)$ // cancel out the last contribution
    $r^-, r^+ \leftarrow SELECTION(r^i)$ // select the critical values r^-, r^+
    $s^i \leftarrow CONTRIBUTION(r^-, r^+, \kappa)$ // compute new contribution
    $\bar{c} \leftarrow UPDATE(r^i, s^i)$ // update $\bar{c}$ to its new value
  endfor
  $\kappa \leftarrow INCREASE(\kappa)$ // κ parameter sets the convergence speed
until no sign changes in $\bar{c}$
for each variable $j \in N$ **do**
  if $\bar{c}_j < 0$ **then** $x_j \leftarrow 1$ **else** $x_j \leftarrow 0$

Figure 4. Serial Optimization Algorithm

r^i represents a local temporary copy of the $\bar{c}$ entries that correspond to constraint *i*, r^- and r^+ are the smallest and the second smallest entries of r^i, also called *critical variables* for a particular constraint, s^i is the contribution of constraint *i* to $\bar{c}$ and $\kappa \in [0,1)$. For each constraint *i* there exists a unique sparse vector s^i, due to the fact that the algorithm requires the cancellation of the previous contribution to $\bar{c}$ before a constraint is iterated again.

4.3. Parallelization Approach

The parallelization of the optimization process must be done within a single iteration in order to retain the convergence characteristics of the method. The first parallelization attempt used a row-wise decomposition approach. Theory and results on this scheme can be found in [12]. A second approach based on the column-wise decomposition of the problem called *variable based decomposition* (VBD)[1] was pursued.

The VBD approach is realized by distributing subsets of variables (columns) to each processor. Each processor is then responsible for a part of $\bar{c}$ and the corresponding part of the *A*-matrix. Some of the operations needed for the constraint update can be conveniently done locally (e.g., *CANCEL*, *UPDATE*), but the *CONTRIBUTION* operation requires communication. Each worker process, first executes in parallel the *SELECTION* operation to find the local minimum values, and then communicates them to the manager process. The manager calculates and broadcasts to the workers the contribution s^i of the current constraint *i* to the reduced cost vector. Lastly, the workers perform the *UPDATE* operation and proceed with the next constraint. The communication involves the transmission of a large number of small messages, which makes the use of an efficient low latency communication network an imperative requirement. The VBD advantage is the small communicated data volume, which is proportional to the number of worker processes and the constraints of the problem. Load balancing is very important for the VBD approach and a simple and effective strategy is to send randomly selected variables to each worker process [26,23].

It can be observed that the main performance issue of the VBD approach is the requirement to perform $O(m)$ synchronization operations per global iteration. To overcome this and based on the fact that constraints without common variables can be iterated independently, such groups should be found [21]. If we consider the constraint dependence graph where the nodes are the problem constraints and the edges connect constraints with common variables, the problem of identifying groups of independent constraints can be solved as a graph coloring problem. All the constraints colored with the same color are independent. The fact that the graph coloring problem is *NP*-hard is not so crucial, since there is no need for an optimal solution but for a reasonable approximation [22,26]. A non-optimal fast graph coloring algorithm based on [18] is used for the creation of the constraint groups. If a constraint set *g* contains only independent constraints, then the *CONTRIBUTION* operation can be performed group-wise for all $|g|$ constraints at once. The *UPDATE* operation can now be delayed till the end of each constraint group. The benefit of this approach is that without changing the convergence characteristics of the algorithm we have managed to reduce the number of the communicated messages and synchronization steps from the total number of constraints to the number of independent constraint groups. This strategy was first done on an SGI Origin 2000 at the Chalmers University of Technology, with a speedup of 7 when 8 processors were used [22,26]. However, it can not be used directly on a conventional network of workstations, due to the high latency of the interconnection network. To make this strategy more viable for NOWs a low latency communication network is needed [6,15] with hardware routers, which allows simultaneous communications of all processors. In addition, an optimized message passing implementation [8] is necessary, which eliminates memory copies and uses an efficient protocol stack.

To overcome the limitations of traditional and inexpensive networks an algorithmic improvement of the previous parallel algorithm was attempted. The variation is based on a "lazy" updating procedure of $\bar{c}$. Only the common variables with the constraint group that will be iterated next (*PARTIAL_UPDATE* operation, see Figure 5) is updated. The reduced cost vector is then fully

updated (*FULL_UPDATE* operation), based on the contributions of the previous constraint group, during the time the manager process calculates the contribution to $\bar{c}$ of the current constraint group. This approach overlaps computation with both communication and previous idle time due to synchronization, given that the full cost vector update takes a significant amount of time. The computation time is slightly increased, and the relaxation may have no benefit at all in the extreme case of a problem with a structure that does not permit the creation of constraint groups with a small number of common variables among them. Figure 5 shows a high level description of the lazy VBD worker algorithm.

Input: A subset of variables $V_k \subseteq N$, cost vector $C(V_k)$, set of constraint groups $G = \{g_1, g_2, \ldots, g_r\}$, where $g_i = \{i \mid i \in M, \forall j \in g_i\ i$ and j are independent$\}$

Worker Algorithm:

```
c̄ ← C(V_k), κ ← 0, ∀ i ∈ M s^i ← 0
repeat
  for each constraint group g_i ∈ G do
    for each constraint i ∈ g_i do
      r^i ← CANCEL( c̄ , s^i)
      r⁻_local, r⁺_local ← SELECTION(r^i) // local values are selected
      B ← BUFFER(r⁻_local, r⁺_local)
    endfor
    SEND(B, manager) // manager computes CONTRIBUTION of g_i
    c̄ ← FULL_UPDATE(s^{gi-1}) // update with the contributions of the previous constraint group
    s^{gi} ← RECEIVE(B, manager) // receive from manager the contribution of group g_i
    c̄ ← PARTIAL_UPDATE(s^{gi}) // do only the necessary updates for next group, leave work for later
  endfor
  κ ← INCREASE( κ )
until no sign changes in c̄
for each variable j ∈ V_k do
  if c̄_j < 0 then x_j ← 1 else x_j ← 0
endfor
```

Figure 5. High-level "lazy" VBD Algorithm

The size of the reduced cost vector $\bar{c}$ for each processor k is $|V_k|$ and the contribution of a constraint group s^{gi}, is the combination of the contributions s^i of all the constraints of group g_i.

4.4. Theoretical Performance Analysis

The modeling of the execution time for a global iteration of the optimization algorithm as a function of the problem size, the number of processors and other algorithm and hardware characteristics is attempted. The analysis assumes that we have balanced distribution of work and an unloaded network with no special structure, on which global operations require P messages. T_P^{iter} is the iteration time on P processors, NZ and m are the number of non-zero entries and the rows in the constraint matrix respectively, t_s and t_w are the communication latency and cost per word and t_c is the time to complete a basic floating-point operation.

Sequential Algorithm

The serial implementation has iteration time

$$T_1^{iter} = \lambda_1 \cdot t_c \cdot NZ + \mu \cdot m \cdot t_c + \lambda_2 \cdot t_c \cdot NZ \cong \lambda \cdot t_c \cdot NZ \quad (4)$$

where λ, λ_1, λ_2, μ are constants. The first term corresponds to the execution time of the operations *CANCEL* and *SELECTION*, the second corresponds to the *CONTRIBUTION* operation, and the third to the *UPDATE* operation. Based on profiling results the first and third phases take over 95% of the total execution time, which is equally divided among them.

Analysis of the "lazy" VBD approach

In the basic VBD approach the first and the third phase of the algorithm is parallelized, having as a trade-off $m+m$ global reduction/broadcast operations. The expected speedup is thus

$$S \cong \frac{P \cdot T_1^{iter}}{2 \cdot P^2 \cdot m \cdot t_s + T_1^{iter}} \quad (5)$$

In the next paragraphs we make a more detailed analysis of the VBD approach with the constraint groups and the lazy update as in theory, and in practice, proved to be a very successful parallelization technique.

For each constraint group the required communication will be the communication time for each worker process to send the local r^- and r^+ values to the manager process and the communication time required by the manager process to send back to the worker processes the computed values.

$$t_{group}^{comm} = 2 \cdot \sum_{k=1}^{P} (t_s + \sum_{i=1}^{q} p_r \cdot t_w) \quad (6)$$

where p_r and p_s is the volume of the communicated data per constraint in words, and q is the number of constraints in the group. In our implementation $p_r = p_s$. If the problem constraints are partitioned in NG independent constraint groups, using a coloring algorithm, the required communication time will be

$$T_{COMM} = \sum_{i=1}^{NG} t_{group}^{comm} = 2 \cdot P \cdot NG \cdot t_s + 2 \cdot P \cdot m \cdot p_r \cdot t_w \quad (7)$$

With the lazy variable update the idle time of the worker processes is minimized because the communication time (T_{COMM}) and the manager computation time tend to overlap with the computation time used by the worker during the update of the cost vector (*FULL_UPDATE* operation). The idle time of the worker processes with the "lazy" variable update can be approximated as

$$T_{IDLE} \cong \max(0, \frac{2 \cdot P \cdot T_{COMM} - T_1^{iter}}{2 \cdot P}) \quad (8)$$

However, a computation overhead T_{OVER} is introduced

$$T_{OVER} = \lambda_2 \cdot t_c \cdot \sum_{i=1}^{NG} \gamma_i \quad (9)$$

where γ_i is the number of common variables between each consequent pair of constraint groups. This is due to the fact that additional calculations have to be done to determine and partially update the reduced cost vector. The expected parallel iteration time would thus be

$$\begin{aligned} T_P^{iter} &= T_{OVER} + T_{PHASE-1} + T_{IDLE} + T_{PHASE-3} \\ &= \lambda_2 \cdot t_c \cdot \sum_{NG} \gamma_i + \frac{T_1^{iter}}{P} + \max(0, \frac{2 \cdot P \cdot T_{COMM} - T_1^{iter}}{2 \cdot P}) \end{aligned} \quad (10)$$

For a reasonable number of processors and for large problem instances, T_1^{iter} is much larger than the required communication time, and the third term of (10) tends to zero, and thus

$$T_P^{iter} = \lambda_2 \cdot t_c \cdot \sum_{NG} \gamma_i + \frac{T_1^{iter}}{P} \quad (11)$$

which makes the expected speedup equal to

$$S \cong \frac{P \cdot T_1^{iter}}{P \cdot \lambda_2 \cdot t_c \cdot \sum_{NG} \gamma_i + T_1^{iter}} \quad (12)$$

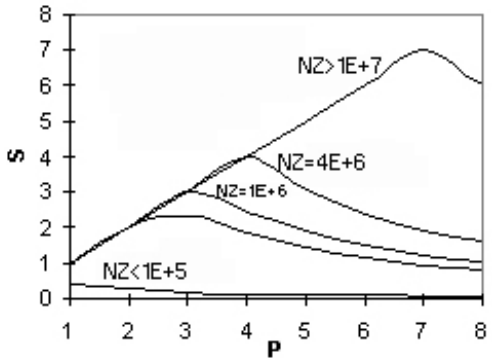

Figure 6. "Lazy" VBD speedup curves for various problem sizes

Based on (12) the expected speedups for various problem sizes of the "lazy" VBD approach are presented in Figure 6. We have assumed an Ethernet based NOW (t_c=0.0293μs, t_s=1500μs, t_w=5μs), 80 constraint groups from the graph coloring and 5% of the non-zeros of a processor common between subsequent constraint groups.

5. Experimental Results

We have measured the performance of the parallel generator and the parallel "lazy" VBD optimizer implementation, using typical crew scheduling problems from Lufthansa. In addition, we report results of the first prototype that integrated the parallel modules in the Carmen system for the same problems. The experiments have been performed on a network of HP715/100 workstations interconnected by standard 10Mbps Ethernet. All workstations used were of almost equal speed (2.89 SPECint95). The implementation of the parallel generator used the PVM message passing library [11] version 4.3, while the implementation of the optimizer used the HP-MPI optimized library [13] version 1.2. PVM provides support for dynamic resource and process control and robustness, used by the parallel generator application. MPI supports asynchronous and non-blocking communication operations, which help the overlap between computation and communication, which is vital for the parallel optimizer. All the programs were written in ANSI C++. The values presented were obtained with exclusive use of the processors and the network.

Name	**dl_kopt**	**dl_splimp**	**dl_gg**	**wk_gg**
legs	1087	946	946	6196
pairings	159073	318938	396908	594560
CPUs	Time in seconds			
1	10860	20760	26460	31380
2	5563	10797	13771	16834
4	2804	5448	7061	8436
6	1892	3686	4536	5338
8	1385	2797	3466	4312
10	1112	2181	2818	3288

Table 1. Results of parallel generator

The parallel generator and optimizer were tested with four different problems of various sizes. In Table 1 we can see the characteristics of these problems and the runtimes of the parallel generator for different number of workstations. The parallel generation time decreases in all cases almost linearly with respect to the number of workstations used.

The output of the generator module then became the input to the optimization filtering module as it is shown in Figure 1. The filtering module attempts to reduce the size

of the constraint matrix by finding equivalent columns and rows and eliminating duplicate or redundant elements. In Table 2 we give the characteristics of the filtered problems and the performance results of the parallel "lazy" VBD optimizer, with NZ representing the number of non-zero elements and s the sparsity ratio of the matrix.

Name	**dl_kopt**	**Dl_splimp**	**dl_gg**	**wk_gg**
Legs	705	641	643	5287
Pairings	156197	316958	393908	590063
NZ	1826456	4074017	4555324	9890583
s (%)	1.65697	1.98101	1.79009	0.51
CPUs	Time in seconds			
1	951.53	1498.13	2763.12	4071
2	634.54	932.14	1493.57	2035.48
3	365.90	739.76	1001.13	1380,53
4	259.32	434.19	752.89	1041.76

Table 2. Results of "lazy" VBD optimizer

The quality of the solution remains as in the sequential execution while the speedup that is achieved is quite significant. Particularly, for the larger problem *lh_wk_gg*, the speedups are excellent because a small number of large constraint groups exist for this problem. Consequently, the granularity of the computation work is coarse-grained, the communication with the manager is sparse and the idle time is close to zero. In addition, the structure of the problem, which is characterized by long and sparse constraints, makes the overlap between consequent constraint groups minimum. This implies that the overhead term is also minimized. From the experimental results it can be concluded that the theoretical analysis does hold and the ability to use networks of existing workstations for this work is validated. For the problems of this experiment the use of more workstations does not improve the execution time. As it can be seen from Figure 6 about eight workstations can be maximally used efficiently for practical crew scheduling problems.

The parallel components have been integrated in a prototype system, coexisting with the sequential Carmen components. These sequential components take 5-15% of the total runtime on the average, depending on the size of the problem. We run the test problems with the prototype system and we report the results in Table 3. The total runtime of the parallel system is the sum of the parallel generation time, the parallel optimization time and the time spent in the sequential components of the system. The execution of the serial Carmen system on an equivalent machine is also reported. We reduced the execution time about five times for the three problems, and four times for the last problem. The last problem is a large weekly scheduling example where the problem initialization time as well as the connection matrix pre-processing is significant which increases even further the proportion of the sequential components.

	Parallel prototype				**Serial**	
name	**Generator (10 CPUs)**	**Optimizer (4 CPUs)**	**Sequential Part**	**Total**	**Total**	**Speedup**
dl_kopt	1112	259.32	909.44	2280.7	12992.6	5.69
dl_slimp	2181	434.19	2225.81	4841.0	24483.9	5.05
dl_gg	2818	752.89	2922.31	6493.2	32145.4	4.95
wk_gg	3288	1041.76	6381.18	10710.	41832.1	3.90

Table 3. Results of the Carmen system with the integrated parallel modules

6. Conclusions and Future Work

In this paper we presented the prevailing methodology for the solution of the crew planning problem and parallel algorithms for the solution of the main steps of this process. The architecture assumed in the paper and in the ESPRIT/HPCN research project PAROS, involves the use of existing interconnected workstations. The idea has been to better utilize the existing infrastructure for the solution of hard and time consuming combinatorial problems that appear in the context of airline crew scheduling.

The parallelization of the generator and the optimizer will give rise to new business advantages for the Carmen System product. Detailed analysis of the various parallelization approaches for both the generator and the optimizer are presented and the experimental results of the best parallel algorithms on a set of real Lufthansa problems is presented. The improved performance of the system can be used to solve larger problems and/or to increase the problem solution quality. The speed and quality of these parallel methods are therefore critical for the overall efficiency of an airline. The demand for such processes increases even further with the ongoing deregulation of the airline operations in Europe.

On a more technical level, an attempt will be made to avoid the generator manager collection of all the pairings produced by the generator workers. This collection is currently performed due to the fact that a global filtering operation must be done. If this global filtering could be done in parallel, there would be no need to collect all the pairings thus reducing the communication expense of the system. Another implication of this could be that the generator workers could in practice be the same with the optimization workers. In addition, the parallelization of the connection matrix creation and preprocessing step before the generation will be also examined because it has become now the system bottleneck. Lastly, the synthesis

of the "lazy" VBD approach with the sub-problem selection approach of [26] will be investigated.

Acknowledgments

We would like to thank D.Ioannidis, D.Koulopoulos, G.Thomopoulos and C.Valouxis for the fruitful talks and the support in the development and testing of the parallel algorithms. D.Wedelin for his effort in helping us understand the innovative ideas behind the optimization algorithm and his support in difficult situations in the development stage. T.Takkula and P.Sanders for their effort in improving the serial implementations of the optimization algorithm and their cooperation in examining alternative parallelization approaches. We would also like to thank C.Hjorring and O.Liljenzin for the close cooperation in integrating the various parallel modules in the Carmen system. Lastly, we would like to thank all the partners of the PAROS ESPRIT project for their input and support.

References

[1] Alefragis, P., C. Goumopoulos, E. Housos, P. Sanders, T. Takkula and D. Wedelin, " Parallel crew scheduling in PAROS ", EuroPar '98, September 1-4, 1998 Southampton, UK, accepted for publication.

[2] Anderson, E., et. al., *OR in the airline industry*, chapter Crew Pairing Optimization, pp. 228-258, Kluwer Academic Publishers, Boston, London, Dordrecht, 1997.

[3] Barutt, J., T. Hall, "Airline crew scheduling - supercomputers and algorithms", *30th Annual Symposium AGIFORS* 1990, 351-358; and *SIAM News* 23 (6), 19-22, November 1990.

[4] Bianco, L., et. al., "A heuristic procedure for the crew rostering problem", *European Journal of Operational Research*, 58 (1992), pp. 272-283.

[5] Bixby, R. E., W. Cook, A. Cox, E. K. Lee, "Parallel Mixed Integer Programming", Center for Research on Parallel Computation Research Monograph CRPC-TR95554, 1995.

[6] Boden, N. J., D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W.-K. Su, "Myrinet: A gigabit per second Local Area Network", *IEEE-Micro*, 15(1):29-36, Feb. 1995.

[7] Caprara, A., M. Fischetti, P. Toth, "A Heuristic Method for the Set Covering Problem", in *Lecture Notes in Computer Science*, pages 72-84, 1996.

[8] Chien, A. et. al., "High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance", *Eighth SIAM Conference on Parallel Processing for Scientific Computing* (PP97); March, 1997.

[9] Chu, H. D., E. Gelman, E. L. Johnson, "Solving large scale crew scheduling problems", *European Journal of Operational Research*, 97 (1997), pp. 260-268.

[10] Fisher, M. L., P. Kedia, "Optimal Solutions of Set Covering/Partitioning Problems using Dual Heuristics", *Management Science* 36 pp. 674-688, 1990.

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

[12] Goumopoulos, C., P. Alefragis, E. Housos, " High Performance Airline Crew-Pairing Optimization ", European Parallel and Distributed Systems Conference (Euro-PDS98), July 1-3, 1998 Vienna, AUSTRIA, accepted for publication.

[13] HP MPI User's Guide, 3rd Edition, DSW-493.

[14] Housos, E., T. Elmroth, "Automatic Subproblem Optimization for Airline Crew Scheduling", *Interfaces* 27:5, September-October 1997, pp. 68-77.

[15] IEEE, Standard for the scalable coherent interface (SCI), 1993. IEEE Std 1596-1992.

[16] Lavoie S., M. Minoux, E. Odier, "A new approach of crew pairing problems by column generation and application to air transport", European Journal of Operational Research, 35 (1988), pp. 45-58.

[17] Marsten, R., R. Subramanian, S. Martin, "RALPH: Crew Planning at Delta Air Lines", Technical Report, Cutting Edge Optimization, 1997.

[18] Mehrotra, A., M. A. Trick, "A clique generation approach to graph coloring", INFORMS Journal of Computing, 8 pp. 344-354, 1996.

[19] Mitra, G., I. Hai, M. T. Hajian, , "A Distributed Processing Algorithm for Solving Integer Programs Using a Cluster of Workstations", *Parallel Computing*, 23(6): 733-753, 1997.

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

[21] PAROS consortium (Chalmers), *D5.3b Parallel optimizer experiment*, Technical Report, May 1997.

[22] PAROS consortium (Chalmers), *D5.4b Optimizer prototype*, Technical Report, May 1997.

[23] PAROS consortium (Univ of Patras), *D7.2 Dependence analysis and task graph generation of the process for parallel execution*, Technical Report, Dec. 1997.

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

[25] Reeves, C. R., *Modern Heuristic Techniques for Combinatorial Problems*, chapter Lagrangian Relaxation, pp. 243-299, McGraw-Hill, 1997.

[26] Sanders, P., T. Takkula, D. Wedelin, "High Performance Integer Optimization for Crew Scheduling", in preparation.

[27] Wedelin, D., "An Algorithm for Large Scale 0-1 Integer Programming with Application to Airline Crew Scheduling", Annals of Operations Research, 57 pp. 283-301, 1995.