Efficient Trip Generation with a Regulation Modeling Language for Airline Crew Scheduling

Christos Goumopoulos[1], Efthymios Housos[2]

[1]Lyseis Ltd AITS, Patras Scientific Park, GR-26500, Greece
[2]Department of Electrical & Computer Engineering, University of Patras, GR-26500, Greece

Abstract. Trip generation is the most time-consuming phase of the crew scheduling process. A large number of trips must be constructed while satisfying a complex set of regulations. In this paper, we present an efficient trip generation method that utilizes effectively a legality checking system in order to reduce the corresponding search space. Special pruning rules are defined using a high-level rule language, which also supports the modeling of the business regulations required in the scheduling process. In addition, the legality checking mechanism has been tuned to perform efficiently in order to cope with the vast amount of the legality checks required by the trip generator. The algorithms are tested as a module for a crew re-scheduling application satisfying the tight response time requirements of a production system. We present experimental results based on problems provided by a major European airline that validate the usefulness and applicability of our work.

1 Introduction

Scheduling and administering people are difficult and time-consuming processes [1]. The situation is further complicated from the fact that the schedules must satisfy intricate operational constraints. Computer applications that perform the scheduling process are of primary importance because of the extremely high cost of human resources. The combinatorial nature and the size of large-scale resource management create the need for the solution of large problems. Although performance improvements in computer hardware and software are happening continuously, airline problems tend to require also faster solution times due in part to the recent high competition that occurred after deregulation of the airline industry.

An important problem in the airline planning process is the construction of legal trips or lines of work that cover the entire airline's flight, at minimal cost [2]. The problem is usually confronted in two phases. The first one refers to the generation of a large number of legal trips, while the second one to the selection of an optimal set of trips. A trip consists of a sequence of flights to be flown by a crewmember that starts and ends at the crewmember's home base. For short haul fleets the lengths of the trips typically range from one to five days (shifts) with up to 25 flights. Each trip has an associated cost and must satisfy a large number of union, company and governmental regulations.

Trip generation is also used in solutions to crew re-scheduling or recovery problem [3]. Namely, during the recovery process, new trips have to be generated in real time in order to deal with disruptions in operations. For reasonable disruptions, the biggest problems involve up to 15,000 trips. This number would increase considerably for large disruptions like snowstorms at major airline hubs. The difficulty of this problem is not the large number of trips to be considered but the need to generate them as fast as possible. An efficient trip generation process can meet this requirement.

In the past, the regulations that define legal trips have been hard coded into the scheduling applications, with the exception of some external parameters. European airlines, however, frequently wish to modify the rules and these modifications often require more than just a parameter change. Consequently, it has been required extensive software changes for the maintenance and addition of new rules. Therefore, a high-level domain specific rule language would be an ideal solution for the expression and management of rules. Two systems that use a special purpose language for the expression and subsequent management of rules are presented in [4] and [5].

The aforementioned rule systems usually present a black box interface to client applications (e.g., trip generator) examining simply if a partial or a complete trip is legal or not. Experimentations have shown that a different interaction with the rule system could speedup the trip generation phase. This is very important, since the trip generation phase takes, depending on the type and the size of the problem, 70-85% of the runtime required for the solution of a crew-scheduling problem.

The rest of the paper is organized as follows. In section 2, we are making a brief anaphora in the DAYSY regulation handling system since this plays a central role in the trip generation process. In section 3, we define the trip generation problem and describe the search algorithm that is used. In section 4, we present algorithms and methods to improve the performance of the trip generation process. Results on the performance improvements achieved are given based on real airline scheduling problems. Supplementary we reference improvements achieved from a related earlier work that involved the use of parallel processing and present combined results. Finally, conclusions and future work are discussed in section 5.

2 Businesses Regulations Modeling - the DAYSY Approach

As partners of the ESPRIT project DAYSY (Day-to-day resource management systems), we were engaged in addressing the problem of stating and managing regulations for scheduling applications involving human resources. We have researched and suggested a new approach that is based on an Object-Oriented meta-model and a special purpose rule language called *DRL* (*DAYSY Rule Language*) [5]. The prototype we have developed was further upgraded with respect to performance and functionality [6, 7] and is currently utilized by the DAYSY resource management system, which is used in production by Lufthansa Airlines.

DRL represents a modeling language in the scheduling problem domain, designed to express the regulations involved in a user friendly and declarative manner. This is achieved by using high-level language constructs with semantics that are close the user terms. The declarative nature of the language allows the complex data manipulation to be performed transparently to the user in runtime. Another strong feature of the language is code reusability that is achieved by a built-in inheritance mechanism.

DRL programs consist of one or more source rule files, each of which contains some of the text of the program written according to the language's specifications. A number of such files can be compiled together to form a rule-set. A rule-set is composed of a set of logically cohesive rules serving the legality requirements of a particular application and exists at both a high-level representation (DRL language) and a low-level one that is used by the *Legality Checking System* (*LCS*). The LCS is the integration of any rule-set with the LCS Kernel providing services such as creation of the aggregation hierarchy, computation of derived property values, checking the legality of composite activities and on-line manipulation of the rule parameters (see Figure 1).

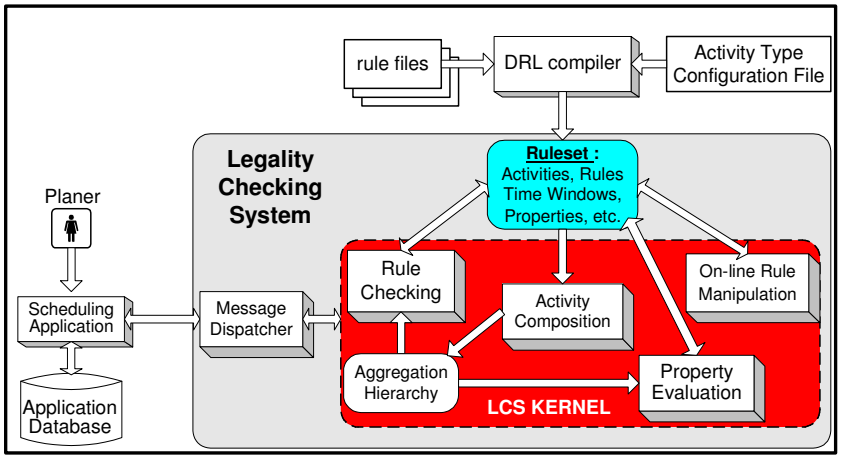


Fig. 1. The DAYSY Legality Checking System (LCS)

A rule compiler transforms, in a first step, the source code in C++ classes and then the ANSI compliant C++ compiler of the target machine translates the intermediate representation to object code that is finally linked to the runtime system. The exploitation of the code optimization capabilities, provided by the C++ compiler, as well as the efficiency of the rule evaluation mechanism described further in this paper, satisfy the performance requirements of demanding applications. In addition, DAYSY LCS allows easy integration with resource management applications through a well-defined C++ API [8]. DAYSY LCS provides an enterprise with a robust rule system for numerous applications that is easy to maintain [9].

3 Trip Generation

3.1 Problem Description

The trip generation process produces a large number of legal trips by combining flights in different combinations. The total number of possible trips depends on the structure of the flight network. The long haul fleets have in general few and long flights and can be examined in an exhaustive manner. A short haul fleet with a reasonable number of planes can produce a huge number of trips if an exhaustive search is attempted. To reduce the number of trips to a reasonable level requires an intelligent generation procedure. The required time to produce all the necessary trips varies between several minutes to several hours, in proportion to the complexity and amount of regulations, the size of the problem and the control parameters of the generation process.

A pre-processed connection matrix that shows the acceptable connections between pairs of flights aids the generation process. The connection matrix represents in mathematical terms a directed acyclic graph among the flights. A node of the graph corresponds to a flight and an edge represents a legal pair-wise connection. Nodes corresponding to flights with a departure station that is a crew home base are identified as *start nodes*. Similarly, nodes corresponding to flights with an arrival station used also as a crew home base are identified as *terminal nodes*. Any path in the graph starting from a start node and ending to a terminal node represents a crew trip, possibly illegal. The possible non-zero elements of the connection matrix for a typical fleet of 1000 flights can be from 10,000 to 100,000.

Formally, let *G(V,E)* represent the search graph for the trip generation process and *HB* the set of crew home bases. Then we define the set of start nodes *S*, the set of terminal nodes *T*, the set of start (terminal) nodes for a specific home base S_{hb} (T_{hb}) and the set of trips *P* (possibly illegal) in graph *G* as follows:

$$S = \{u \in V : departure\_airport(u) \in HB\}$$

$$T = \{v \in V : arrival\_airport(v) \in HB\}$$

$$S_{hb} = \{u \in V : departure\_airport(u) = hb \ \ and \ \ hb \in HB\} \subseteq S$$

$$T_{hb} = \{v \in V : arrival\_airport(v) = hb \ \ and \ \ hb \in HB\} \subseteq T$$

$$P = \{u \mapsto v : u \in S \ and \ v \in T \ and \ departure\_airport(u) = arrival\_airport(v)\}$$

3.2 Search Algorithm

The most practical and at the same time efficient algorithm, with respect to memory resources, for the trip generation process is the depth-first-search (Figure 2) in the search graph. The search always begins from the start nodes of the graph. Every start node is the root of a search tree that will be examined by the trip generation algorithm. The search algorithm traverses the tree until a regulation violates. In every step of the depth-first-search, the created sequence is checked for legality. In case of a rule violation the algorithm backtracks to the previous node and the next branch is examined. The generator search is limited also by a maximum number of branches considered in each node of the search graph.

```
Input: G(V,E) the search graph, S the set of start nodes, R the regula-
tions, SW(p) search width as a function of the working days covered by
p, where p denotes a sequence of nodes
```

```
Algorithm:

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 CHECK(p, R) then
      if p∈P then OUTPUT(p)
      while (SW(p) is not violated) do
         r ← NEXT_CONNECTION(node,E(G))
         SEARCH(r)
      endwhile
   endif
   p ← REMOVE(node)
endprocedure
```

Fig. 2. Trip generation algorithm

Figure 2 shows the trip generation algorithm. The *GENERATE* procedure calls for each start node the *SEARCH* procedure. The SEARCH procedure implements the depth-first-search algorithm for the search tree defined for a specific start node. The functions *ADD*, *CHECK* and *REMOVE* are part of the legality checking application-programming interface.

4 Efficient Trip Generation

4.1 Legality Checking System (LCS)

The efficiency of the trip generation process depends heavily on the performance of the LCS since the rules are checked very many times. A rule-set may contain a large number of rules and

in that case the order of a rule evaluation is very important. The prototype has been using an algorithm that was based on a static priorities policy. A priority value could be defined for each constraint in the rule-set by the rule author. The higher the priority is the earlier the rule will be checked. However, the static priorities scheme was not efficient enough to meet our requirements. We should have known a priori the statistics of the rule violations for a specific problem and specific rule-set in order to select the best priority values.

We have changed the legality-checking algorithm to use a dynamic priorities policy. The algorithm attempts to minimize the number of rules checked, each time a violation happens. The priorities that were defined by the user at compile time are being adapted in runtime so that rules violating more frequently are raising their priority values. The algorithm is based on a multiple queue structure. Figure 3 shows the legality-checking algorithm equipped with the dynamic priorities policy.

```
procedure CheckLegality
   p ← MAXPRI;
   while ( p >= 1 ) do
       if ( Queue(p) not empty ) then
              rule ← get.Queue(p);
       else p--;
       if ( (rule.execution violates)) then
              rule.violation_counter++;
              Terminate;
   endwhile
   if ( timer1 expires)
       call PriorityAging;
   if ( timer2 expires)
       call PriorityDemotion;
endprocedure
procedure PriorityAging
   for all rule in ruleset do
       rule.priority ← rule.priority+floor(rule.violation_counter/
                                          PRIORITY_AGING_LIMIT);
endprocedure
procedure PriorityDemotion
   for all rule in ruleset do
       rule.priority ← floor(rule.priority / 2)
endprocedure
```

Fig. 3. Legality checking algorithm with dynamic priorities policy

The main characteristics of the algorithm are:

- It looks for a queue that is not empty and selects the first rule.
- The legality of the rule is checked. If it is violated the algorithm terminates after updating the corresponding counter for the specific rule. If there is not a violation, the next rule is examined.
- Periodically an adaptation of the rule priorities is attempted by calling the procedure *PriorityAging*. The priority of a rule is increased taking into account the rule violation counter and the parameter PRIORITY_AGING_LIMIT.
- Periodically a demotion of all the priority values happens for accounting purposes by calling the procedure *PriorityDemotion*.

The evaluation of the LCS improvements shows an upgrading of the speed of the legality checking mechanism up to 2 times with respect to the original prototype. These performance improvements allow for the use of the LCS system in automatic decision support tools that require high-speed legality services.

4.2 Enhanced Search Graph

In an attempt to improve, further the performance of the trip generator we have implemented a method that tries to reduce the search space of legal trips by enhancing the search graph with auxiliary information and exploiting, in advance, the services of the legality checking system. The method involves the definition of special pruning rules and scheduling domain properties that can be examined during the traversal of the search graph.

By profiling analysis, it was detected that in several cases the total number of the legality checks performed was excessively larger than the number of the generated trips. This was indicating a frequent trip searching activity in unproductive areas of the search graph. In other words, partial trips in those areas were never developing to complete legal trips. Thereby, the main idea of the method is to predict as soon as possible, the unproductive branches of the search graph and thus to save the unnecessary work and execution time. This method resembles the branch and bound technique that is commonly used to solving integer linear programming problems. In our case, a limit is calculated in every branch of the search graph before the execution of the search algorithm. This auxiliary information is exploited during the traversal of the graph through a specified pruning rule.

We have had first to answer the question of what kind of a limit to calculate in every node of the graph. Given that in the airline domain there are rules that restrict properties of the trip activity to a maximum/minimum value, it is possible to calculate for each node a limit related to these rules. Examples of such rules are the maximum flight and/or work time, the minimum rest time and the maximum number of calendar days allowed in a trip. In general, any constrained property of the trip activity that can be interpreted as available resource running out as a partial trip develops to a complete trip could be used to enhancing the search graph. For instance, if the sum of the accumulated work time of a partial trip and the minimum work time to complete that partial trip is greater than the allowed, it is concluded that this partial trip will never develop to a complete legal trip.

To enhance the search graph with the auxiliary information we apply an extension of the well-known Dijkstra algorithm [10]. Dijkstra's algorithm applies to a directed graph with positive weights and calculates the shortest path from a source node to every other node in the graph. In our case, the weight of every edge in the graph equals to a domain-specific calculated property value and we consider as source nodes every terminal node in the graph.

Figure 4 shows Dijkstra's algorithm with the necessary extensions (marked with the symbol >) for calculating the limit values in every node of the graph. A new procedure with name *DRIVER* is introduced calling the procedure *BOUND* for each terminal node of a specific home base. The functions *ADD*, *REMOVE* and *EVALUATE* are part of the legality checking application-programming interface. In particular, the function *EVALUATE* calculates a property value for each node in the graph. The *PROPERTY* parameter specifies the identifier of the calculation rule that is triggered for the evaluation. Also, for each node $v \in V$ of the graph *G(V, E)* we keep the structure d[v] for storing the calculated value and the structure $\pi[v]$ for storing the precedent node of v in order to keep the right track of the sequence p that is passed to the legality checking system for the evaluations.

```
Input: G(V, E) the search graph, T_hb the set of terminal nodes for a
specific home base. p stores a sequence of nodes.
 procedure INITIALIZE(G, s, p)
   for each node v in V[G] do
 >   π[v] ← NULL
   endfor
 > p ← ADD(s)
 > d[s] ← EVALUATE(p, PROPERTY)
 endprocedure
 procedure RELAX(u, v, p)
 > p ← ADD(v)
 > temp ← EVALUATE(p, PROPERTY)
   if d[v] > temp then
     d[v] ← temp
 >   π[v] ← u
   endif
 > p ← REMOVE(v)
 endprocedure
 >procedure DRIVER(G, T_hb)
   for each node v in V[G] do
     d[v] ← infinity
   endfor
   for each node s in T_hb do
     BOUND(G, s)
   endfor
 endprocedure

 procedure BOUND(G, s)
   INITIALIZE(G, s, p)
   D ← {empty set}
   Q ← V[G]
   while (Q not empty) do
     u ← EXTRACT-MIN(Q)
     D ← D ⋃ {u}
 >   p ← ADD(π(u))
     do
        v ← NEXT_BWD_CONNECTION(u, E(G))
        RELAX(u,v, p)
     until (all connections have been used)
    endwhile
 >  for each node k in p do
      p ← REMOVE(k)
    endfor
 endprocedure
```

Fig. 4. Dijkstra's algorithm used to enhancing the search graph

As mentioned before, a constrained property that can be used for the pruning of the search graph is the work time allowed in a trip. The PROPERTY (`work_time`) that is calculated in every node of the graph has the following top-down definition in DAYSY Rule Language notation:

```
PROPERTY work_time OF crr
       RULE: (sum over shift (duty_end - duty_begin));
END
PROPERTY duty_end OF shift
       RULE: (of last leg arrival) + TERMINATING_ACTIVITY;
END
PROPERTY duty_begin OF shift
       RULE: (of first leg departure) - PREPARING_ACTIVITY;
END
```

A trip (`crr` activity) consists of one or more shifts. The aggregation operator '`sum`' adds the work time of all the shifts. The keywords '`arrival`' and '`departure`' are data providers while the reserved words '`last`'/ '`first`' are special reference operators of the rule language.

The pruning rule that must be defined in order to exploit the enhanced search graph will have the following form in DAYSY Rule Language notation:

```
CONSTRAINT MRA_pruning_rule OF leg
  PRIORITY: MAXSETPRI;
  RULE:(of parent crr (work_time)) + bound <= MAX_CRR_WORK_TIME;
END
```

The pruning rule is defined for every flight (leg activity) and is checked with the highest priority. The 'work_time' property accumulates the work hours for the so-far created trip (parent crr activity). The keyword 'bound' is a primitive property of the leg activity defined in the rule language. This keyword serves as an interface between the high-level rule modelling process and the low-level calculations performed by Dijkstra's algorithm. The parameter MAX_CRR_WORK_TIME specifies the maximum work time allowed in a trip. Figure 5 depicts a simplified application of the above pruning rule. A partial trip has been constructed with 20 hours of accumulated work time. The minimum work time values to complete a trip (i.e., to return to the home base) for every unexplored node are shown in the figure. We assume also that the maximum work time allowed in a trip is 36 hours. As it is shown, the rule prunes two of the three possible branches for the partial trip expansion and thus speedups the trip generation process.

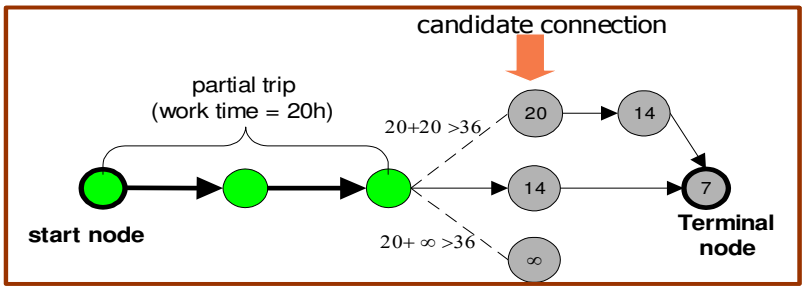


Fig. 5. Example of a pruning rule

The performance improvement of the trip generation process with the enhanced search graph depends on the choice of the PROPERTY that is calculated and the frequency of the pruning rule violation. We have seen an improvement of up to 230% in the speed of the trip generator for some problems. In some other problems, mostly when large trips are allowed, it was noticed that the rule has difficulties to prune branches early in the search process. Thus, in this case the method is less effective. Finally, the enhancing overhead for a typical problem with 10^3 nodes and 10^5 edges is approximately 0.1% of the total generation time.

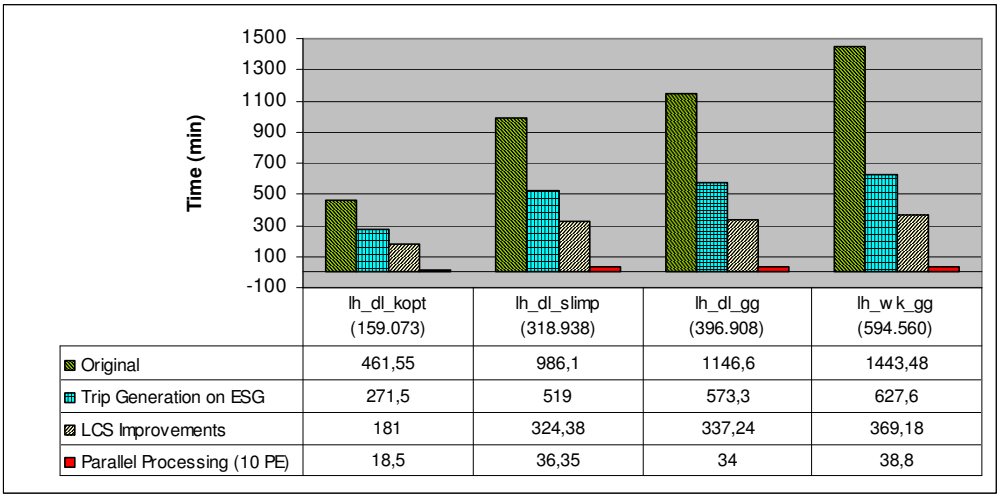
4.3 Parallel Processing

As partners of the European ESPRIT/HPCN project PAROS (Parallel Large Scale Automatic Crew Scheduling) we were faced with the challenge of improving the performance and extending the functionality of an automatic crew scheduling process with the use of high performance computing and modeling techniques on a network of workstations (NOW). Lufthansa German Airlines was the coordinating partner of the project and supplied important large problems and optimization requirements. In [11, 12] we have presented successful parallel algorithms and techniques that were developed to solve the airline crew scheduling problem on the NOW architecture. In particular, the parallel trip generator component of the crew scheduling process achieves a linear speedup on the number of processors and it can be efficiently scaled to a large number of processors.

4.4 Combined Results

The performance improvements on the serial trip generation phase presented in this paper have a multiplicative effect on the speedup achieved when parallel processing is used. Figure 6

shows these performance improvements when algorithmic enhancements and parallel processing on a network of 10 workstations are combined. Four typical Lufthansa problems of various sizes (we note in parenthesis the number of trips generated) and the Lufthansa production rule-set consisting of 55 rules were used for the experiments. The legality checking mechanism performance improvements and the trip generation with the use of pruning rules on the enhanced search graph yield an average speed gain of 3.5 times with respect to the original time. This means that the trip generation runs approximately 35 times faster when 10 processors are used.



	lh_dl_kopt (159.073)	lh_dl_slimp (318.938)	lh_dl_gg (396.908)	lh_w k_gg (594.560)
Original	461,55	986,1	1146,6	1443,48
Trip Generation on ESG	271,5	519	573,3	627,6
LCS Improvements	181	324,38	337,24	369,18
Parallel Processing (10 PE)	18,5	36,35	34	38,8

Fig. 6. Performance improvements combining algorithmic enhancements and parallel processing

5 Conclusions

An efficient trip generation component allows for the better confrontation of time-consuming problems like the airline crew scheduling and day-to-day rescheduling problems. We have presented improvements to a prototype trip generator. A legality checking system is utilized effectively in order to reduce the trip search space. Further, the legality checking mechanism has been tuned to perform efficiently in order to cope with the vast amount of the legality checks required by the trip generator. The performance improvements gained from this work allow for the use of the system in automatic decision support tools in the scheduling/ rescheduling domain.

The improvements in performance translate immediately in significant benefits for the airline. 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.

Profiling on a typical level rule-set revealed that 40-60% of the rule evaluation time is spent on the evaluation of constraints expressed over moving time windows. This is a basic feature of the DAYSY rule handling system. Future work includes an investigation to improve the time window handling mechanism. An early study of applying range search techniques, often used in computational geometry algorithms, and a pre-processing phase to identify the critical areas

where a constraint is more probable to fail, shows that is possible to achieve a significant reduction in the computation time.

References

1. Nanda, R., and J. Browne, *Introduction to Employee Scheduling*, John Wiley & Sons, New York, June 1992.
2. Anderson, E., Housos, E., Kohl N., and Wedelin D., Crew Pairing Optimization, *Operations Research in the airline industry*, G. Yu (editor) Kluwer Academic Publishers, Boston, London, Dordrecht, pp. 228-258, 1997.
3. Letovsky, L., Airline Operations Recovery: An Optimization Approach, Ph.D. thesis, Georgia Institute of Technology, Atlanta, 1997.
4. Carmen Systems AB, Carmen PAC 5.0 – User's Reference Manual, Carmen Systems AB, Gothenburg, Sweden, November 1996.
5. Thrampoulidis, K., Goumopoulos, C., and Housos, E., Rule Handling in the day-to-day Resource Management problem: an Object-Oriented approach, *Information and Software Technology*, vol 39, pp. 185-193, Elsevier Science, 1997.
6. Alefragis, P. and Housos, E., Performance Improvements of the DAYSY Legality Checking System for Real Time Rescheduling Applications, *Proc. of the fifth International Conference of the Decision Sciences Institute (DSI'99)*, vol. I, pp. 216-218, Athens, Greece, July 4-7, 1999.
7. Goumopoulos C., Automatic crew scheduling involving high-level modeling of the regulations and use of parallel/distributed processing, Ph.D. thesis, Electrical and Computer Engineering Department, University of Patras, Greece, June 2000.
8. Goumopoulos C., and Alefragis, P., *Legality Checker C++ Application Programming Interface version 1.1*, ESPRIT PROJECT EP8402 – DAYSY, Patras, Greece, Jan 1997.
9. Goumopoulos, C., Alefragis, P., Thrampoulidis, K., and Housos E., A Generic Legality Checker and Attribute Evaluator for a Distributed Enterprise Environment, *Proc. of* the *third IEEE International Symposium on Computers and Communications (ISCC'98)*, IEEE CS PR08538, pp. 286-292, Athens, Greece, June 30 - July 2, 1998.
10. Cormen, TH., Leiserson, CE., and Rivest, RL., *Introduction to Algorithms*, MIT Press, Cambridge, Mass, pp. 520-527, 1990.
11. Alefragis, P., Goumopoulos C., Housos E., Sanders, P., Takkula, T., and Wedelin, D., Parallel Crew Scheduling in PAROS, *Proc. of the fifth International Euro-Par'98*, LNCS 1470, pp. 1104-1113, Southampton, England, September 1-4 1998.
12. Goumopoulos C., Alefragis, P., and Housos E., Parallel Algorithms for Airline Crew Planning on Networks of Workstations, *Proc. of* the *27th International Conference on Parallel Processing (ICPP'98)*, IEEE CS PR08650, pp. 70-78, Minneapolis Minnesota, USA, August 10-14 1998.