

CICLIC: A Tool to Generate Feasible Cyclic Schedules

José Yépez, Josep Guardia, Manel Velasco, Jordi Ayza, Rosa Castañé, Pau Martí and Josep M. Fuertes
Automatic Control and Computer Engineering department, Technical University of Catalonia
C/Pau Gargallo 5, 08028 Barcelona, Spain
{jose.yepez, josep.guardia, manel.velasco, jordi.ayza, rosa.castane, pau.marti, josep.m.fuertes}@upc.edu

Summary

A common way of implementing periodic real-time systems is by means of a cyclic scheduler. This method of scheduler is characterized by having a high degree of determinism, a high degree of predictability, a high degree of reliability, and implementation simplicity. Also, it is a very well-known method broadly used in industrial environments. The main problem this type of scheduler may have is the lack of flexibility, since any change in the group of tasks or in its temporary characteristics forces it to do repeat the execution plan. In solving these problems, it is necessary, and most of the time, essential, to have tools that may help system designers to create a cyclic scheduler automatically. In this paper, a tool having this function, and named CICLIC is described. The design mechanism is based on algorithms of exhaustive search that use heuristic rules in optimizing searching paths. To illustrate the use of CICLIC some examples are shown.

Key Words: cyclic scheduler, real-time systems, simulation.

1 INTRODUCTION

One of the most important characteristics of a real-time system is that their correct operation depends on both the setting of adequate exits and the completeness of the task achieved within the interval of required time [16]. This requirement leads to one of the fundamental problems in the design of real-time systems: the scheduling of execution time producing responses within a guaranteed amount of time.

The general problem of automating the generation of a feasible scheduling in a group of tasks is a NP-Hard problem [8], which means that in the worst case an exponential amount of work appears necessary to determine the existence of a feasible schedule. One of the approaches to build real-time systems consists on calculating acceptable scheduling before starting the system, i.e., off-line. Then, real-time dispatcher executes the tasks according to an established execution plan.

The classic approach of an off-line real-time

scheduler is made by a cyclic scheduler. The cyclic scheduler [1, 7] follows an iterative procedure that allows explicit multiplexation of a group of tasks on a processor. The cyclic schedules are defined by scheduling tables or execution plans, which are built by the group of tasks and their temporary restrictions. Each table input has a code segment associated to each plan task. The processor executes each segment code of the table. Once all table inputs have been executed, the processor returns to the first table input and the process is then repeated. Periodical clock interruptions are produced, to keep cyclic scheduling synchronized with the time of the application.

In particular, the standard scheduling table contains a main plan which defines task sequence to be executed within a fixed period of time, also called main cycle. The main plan is divided into one or more secondary plans, where task sequence to be executed within a fixed period of time or secondary cycle is included. In general, secondary plans of the same main plan will contain sequences of different tasks, depending on their temporary characteristics. In fact, each secondary plan has one or more frames. A part or a complete task is then executed within a frame, according to the secondary plan sequence.

When checking timing constraints of concurrent tasks, designer should verify first such constraints are met. Since 1980, many models, methods, and tools have been proposed to check if a real-time system fulfills its requirements [2, 11]. This theory helps any system designer to study and predict the timing behavior of real-time tasks through scheduling simulation and feasibility tests. Scheduling simulation requires first to compute a scheduling on a given time interval and then, to look for timing properties in this computed scheduling. On the contrary, feasibility tests allow the designer to study a set of real time tasks without computing scheduling.

The first contributions to real-time scheduling theory appeared 30 years ago [11]. This theory has been widely extended to cope with many application requirements [10, 18], and has been successfully used in many projects [13]. However, only few tools were proposed to help system designers to generate these results automatically. Here, some of them are briefly described:

Rapid-RMA [15] and TimeSys [14] seem to be the most complete tools. These tools provide most of the classic scheduling algorithms and feasibility tests. They can be connected to different programs, middleware, operating systems. However they are not free.

YASA [3] and the tool from the Université Libre de Bruxelles [6] focus on scheduling simulation only. They do not provide feasibility tests.

MAST [9] is based on Ada framework. It provides some feasible tests and simulation services for fixed priority schedulers.

CHEDDAR [5] provides a free and flexible work environment, which implements most methods of the classic planning theory. It, however, does not include cycling scheduling.

Most of these tools do not determine execution plans automatically for cycling scheduling, with the exception of Rapid-RMA. This is largely due to the nature of the problem to be solved and the absence of algorithms that could generate execution plans automatically within a reasonable period of time.

In this paper, a tool called CICLIC is described. This tool may be seen as a support for real-time system designers to build and maintain cyclic scheduling. The development of this tool has been considered because of the lack of flexible tools for cycling scheduling. It can be used in whatever application where an offline scheduler is used, as p.e. automotive, real time communications, etc. CICLIC design mechanism is based on exhaustive search algorithms using heuristic rules to optimize the search path. CICLIC is a portable, reliable and easy tool.

This paper is organized as follows. In Section 2 the proposed algorithm is described. Here, characteristics of cycling planning are first described, by showing the complexity of the problem and the optimization in the search of a feasible planning. In the Section 3, CICLIC characteristics are described. In Section 4, two examples are given to show how CICLIC are used. Finally, Conclusions and future work orientation are offered in Section 5.

2 CYCLIC SCHEDULER

Cyclic schedulers execute tasks in sequence according to some scheduling tables or execution plans. These tables or plans are set before starting the system. In building a plan, the extent of cycling periods (main cycle, secondary cycle) should be first determined. Next, task executions should be assigned to frames until a planning of tasks timing constraints be completely achieved. In this way, scheduling of secondary cycle clocks is used. There, a clock pulse occurs each time a secondary cycle is over, so that a cycling planning may be synchronized.

Time length of main and secondary cycles should meet some requirements to get an acceptable scheduling. These conditions are next described [7].

2.1 CHARACTERIZATION OF CYCLING PERIODS

Given a group of n periodic tasks represented by the group $\{(C_1, T_1, D_1), (C_2, T_2, D_2), \dots, (C_n, T_n, D_n)\}$, where C is the execution time, T the period and D the deadline. In accordance with these definitions assume the relationship $C \leq D \leq T$.

Time length of the main cycle M , also called hyperperiod, is the least common multiple of the task periods,

$$M = lcm(T_i) \quad (1)$$

Some acceptable solutions may be considered in determining the extent of m secondary cycle. If there is only on timer controlling secondary cycles length, the value of m should meet the following conditions [1, 7]:

- a. m should be smaller than or equal as the execution term of any task:

$$\forall i : m \leq D_i \quad (2)$$

- b. m should be larger than or equal to the largest compute time given by the tasks:

$$\forall i : m \geq \max(C_i) \quad (3)$$

- c. m should be a submultiple of the time length of the main cycle M :

$$\exists k : M = k.m \quad (4)$$

- d. To get a complete secondary cycle between the onset and conclusion of each task, m should meet:

$$\forall i : m + (m - \gcd(m, T_i)) \leq D_i \quad (5)$$

In case no value can verify the group of previous conditions, the group of tasks cannot be scheduled with a cyclic scheduler.

2.2 TASK INSERTION IN SECONDARY CYCLES

From above conditions (2), (3), (4) and (5), secondary cycle m values may be obtained. Since time length of M is fixed, the chosen m value will get $n_{cs} = M/m$ secondary cycles in the main plan.

Therefore, each main plan is composed by n_{cs} secondary plans. On the other hand, each main plan includes n_{ei} executions of each task τ_i .

$$n_{ei} = \frac{M}{T_i} \quad (6)$$

So, each secondary plan is formed by a frame

sequence: $\sigma = \{\tau_{i_k}, \tau_{j_l}, \dots, \tau_{i_s}\}$, being each one associated with a task execution.

From (5), it can be inferred that the k^{th} execution, $\forall k \in [1 \dots n_{e_i}]$ task execution i , characterized by (C_i, T_i, D_i) could be included in the j^{th} secondary cycles, $\forall j \in [1 \dots n_{c_s}]$ if:

$$(k-1) \times T_i \leq (j-1) \times m \leq (k-1) \times T_i + D_i - m \quad (7)$$

To include k^{th} execution of τ_{i_k}, τ_{j_k} , in $\sigma(m_j)$, guaranteeing timing constraints, m_j should meet (7), and the following condition:

$$C_i \leq m - \sum_{\forall l | \tau_l \in \sigma(m_j)} C_l \quad (8)$$

This is to be done in such a way that free time in this secondary cycle should be left in executing the task.

2.3 IN SEARCH OF A FEASIBLE SCHEDULING

An acceptable scheduler allows the group of tasks be executed to fill with its timing requirements. The problem of establishing an acceptable scheduling, back-tracking algorithms can be used.

Back-tracking algorithms look for a solution searching the space solutions of the problem. The search is simplified by organizing the space of solutions on a tree-diagram order. In this case, the root of this tree Π_0 is the acceptable scheduler for 0 tasks. Each node Π_i of this tree contains a partial solution to the problem of scheduling i tasks, where i is the node depth. Each node Π_i has h possible successors. These successors comprise all the ways the $n-1$ tasks without schedule may be included in scheduling. The space of solutions is defined by all the paths from the node root to the terminal nodes. Each tree node of the space of solutions represents a state of the problem. This is one reason the tree-diagram order is also called a state space tree.

Back-tracking algorithms search in depth the state space tree until it finds an acceptable solution for the problem. This will generate all the possible states of the problem. To avoid generating all these states a pruning function is used to identify those nodes that do not lead to acceptable solutions without generating the corresponding sub-tree. The algorithm concludes when a node Π_i of n depth is reached, showing thus a solution. It also concludes when it is not possible to expand any node, supposing no solution may be found and no feasible cycling scheduling for the group of tasks.

2.4 COMPLEXITY OF THE PROBLEM

The space of solutions is formed by all the ways of including the n tasks in the scheduling, enclosed by

the following equation [7]:

$$\prod_{i=1}^n \binom{n_{cs}}{n_{e_i}} = \frac{(n_{cs}!)^n}{\prod_{i=1}^n n_{e_i} \times (n_{cs} - n_{e_i})!} \quad (9)$$

From this equation, the complexity of the problem is increased as the number of tasks and secondary cycles also increase. Therefore, if there are several values of m verifying the equations (2), (3), (4) and (5), the higher values should be chosen first to find out a feasible scheduling.

2.5 OPTIMIZATION IN THE SEARCH OF A FEASIBLE SCHEDULING

In the worst case, the tree of the space of solutions should be searched completely to find out an acceptable planning for the group of the n tasks or to assure that it does not exist. However, the complexity of the space of solutions (9) is quite large. There are different methods that may be used to build cyclic scheduling [4], such as heuristic search, stochastic evolution, and genetic algorithms. Specifically, in this study, some heuristic rules have been used because solutions may be found easily.

The rules here used allow the best option for expanding a node through one of its successors. It is expected this path may be used to get a node of depth n , generating a reduced number of tree nodes and, therefore, the complexity of the problem.

In this case, the criteria selected to choose the task are:

1. To add the task with the shortest execution time among the remaining ones.
2. To add the task with the shortest period among those waiting for scheduling.
3. To add the task with the longest computing time among those waiting for scheduling.

Tasks are chosen in the developed algorithm according to the first rule order; the other two rules left will be used once similar response time tasks appear. These criteria allow choosing both less feasible tasks and tasks with greater difficult tasks for scheduling. Thus, by the use of these criteria faster solutions may be found instead of using more arbitrary tasks.

Once a task τ_i is chosen, its n_{e_i} executions should be included in the secondary cycle. In general, for each execution τ_{ij} more than one secondary cycle will be there where it can be included. The problem of adding tasks to a partial scheduling may be compared with the process of bin packing [8].

In choosing a secondary cycle where the execution τ_{ij} will be included, heuristic rules are applied and used

in the problem of bin packing:

1. To include the task in the first secondary cycle where computation time is higher than or equal to the highest computation time of the task. This is equal to bin packing where first fits.
2. To include the task in the last secondary cycle where its timing requirements should be fulfilled. This is equal to bin packing where the last fits.
3. To insert the task in the secondary cycle where lesser free computation time may be left and timing requirements be fulfilled. This is equal to bin pack where best fits.

2.6 ALGORITHM

The figure 1 shows the algorithm flow diagram where CICLIC is based to determine and to simulate cyclic scheduling.

This algorithm is formed by several processes. The first one, based on timing specifications of the group of tasks, checks schedulability conditions (fig1:B) through the rules given in Section 2.1. If these conditions are fulfilled, both the main and secondary cycles of the scheduling are then determined (fig1:C). Secondly, the group of tasks is ordered according to heuristic rules determined in Section 2.5. Also, frames and executions used in each task is determined (fig1:D). Thirdly, a feasible scheduling is expected to be found by inserting task executions into the secondary cycles. This process should meet time requirements (fig1:E). Scheduling is built recursively (fig1:F) until a feasible cyclic scheduling is found (fig1:G) or until the whole group of tasks cannot be inserted. We may then think that the problem has no solution (fig1:H).

3 THE CICLIC TOOL

CICLIC is a Java™ tool used in the design and development of cyclic scheduling on real-time systems based on a processor which gives some analysis and simulation utilities. Through CICLIC cyclic scheduling may be established for later implementation over any processor and real-time operating system.

CICLIC users may specify systems tasks. All specifications are read from the filing system provided by the used and compiled in an internal setting. This description may be easily seen and modified through Graphic User Interfaces (GUIs).

The user may assign a color to each task, and then determine and verify relevant events, measure time interval and check timing constraints completion. After reading the system features, analysis and simulation utilities can be used. Through windows-

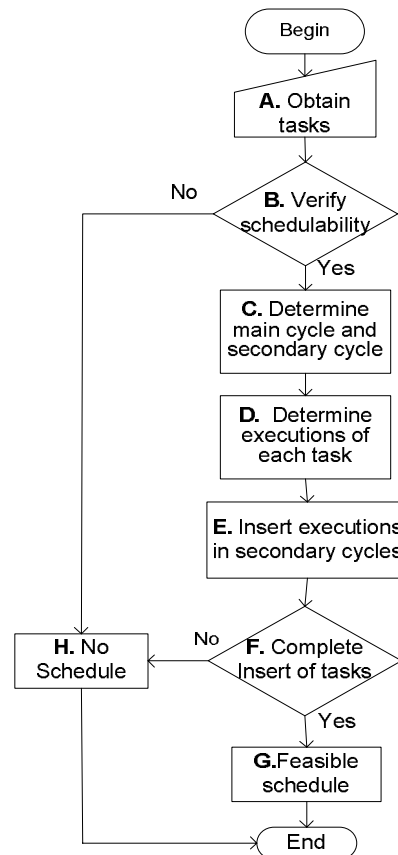


Figure 1: Diagram flow of algorithm of the tool CICLIC based interface, CICLIC shows:

- The group of tasks
- Scheduling analysis
- C Code of the found scheduling
- Scheduling simulation

Thus, CICLIC reduces update task impact and/or modification, facilitating thus, its completion.

3.1 TASK SPECIFICATION

Users should specify each task attribute: Task period (T), deadline of the task (D), expected time of task execution (C) and the name of task description.

3.2 SCHEDULABILITY ANALYSIS

CICLIC checks schedulability conditions for cyclic scheduling (Section 2.1) and determines the use factor of the CPU. If no value for these conditions is found, the group of tasks cannot be scheduled through a cyclic scheduler.

3.3 C CODE

CICLIC generate the set of instructions in C code, according to the given scheduling.

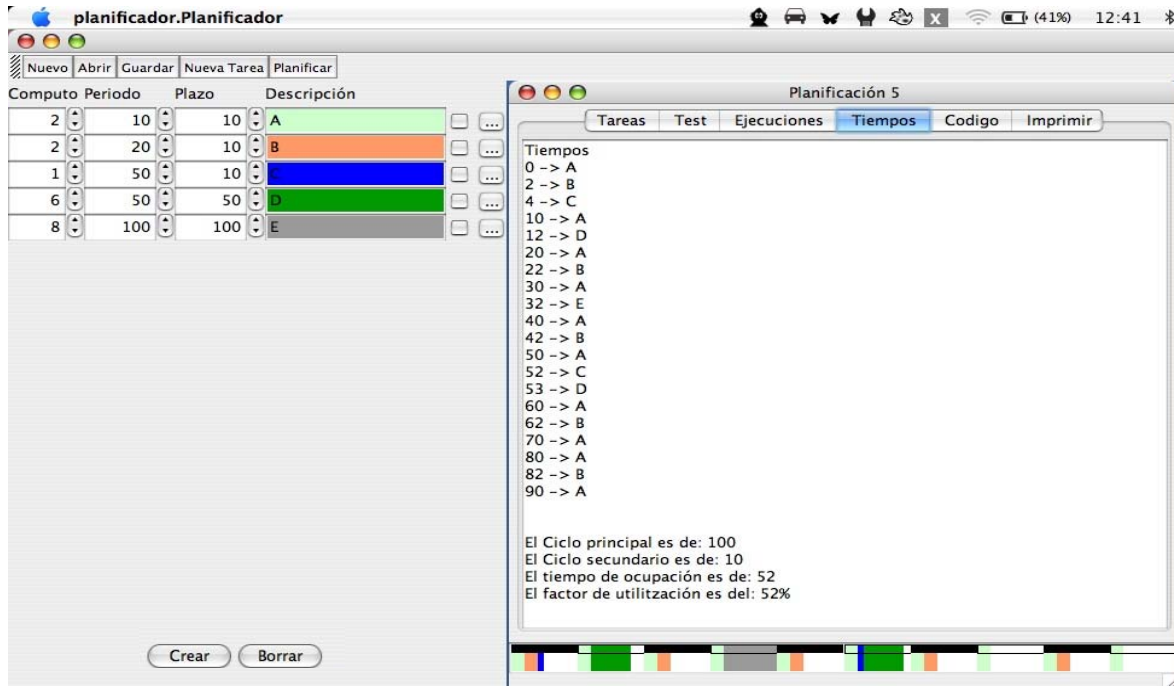


Figure 2: Graphic interface of CICLIC

3.4 GUI OUTPUT WINDOWS

CICLIC graphic user interface has the following advantages:

1. The users may insert, see, modify, and save the description of the group of tasks.
2. It performs and show scheduling analysis.
3. It shows the CPU load.
4. It produces and show simulations of feasible cyclic scheduling when present.

Examples of these windows are shown in Figures 2, 3, and 4

4 STUDY CASE

In this section, CICLIC is shown in the analysis and design of cyclic schedulers on real-time systems. The first study case deals with a Volvo Construction Equipment (VCE) [17]. VCE is normally based on static schedulers, due to the configuration of its control systems for heavy machinery and equipment (trucks, articulated haulers, backhoes, etc). We will be based on a simplified version of one VCE built systems [12]. In particular, the group of tasks shown in Table 1 is considered.

Figure 2 shows CICLIC's GUI where the group of tasks is shown together with scheduling analysis results, time length of both the main and secondary cycles, and CPU utilization factor.

Figure 3 shows the simulation of the cyclic scheduler, where starting and final task execution is clearly observed. Each task has a color. White stripes

show CPU free time. The beginning and final of each secondary cycle is indicated by a black box that appears/disappears in the upper section of simulation.

The second study case deals with a simplification of a car control system, whose group of tasks are shown in Table 2.

Task	T_i	C_i	D_i
A	10	2	10
B	20	2	10
C	50	1	10
D	50	6	50
E	100	8	100

Table 1: Set of tasks VCE system.

Task	T_i	C_i	D_i
A: Speedometer	20	4	20
B: ABS control	40	10	40
C: Fuel injection	80	16	80

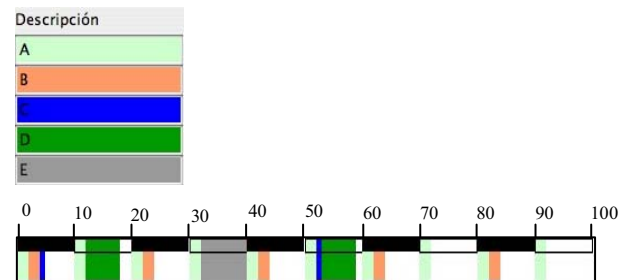


Figure 3: Cyclic scheduler of the group of tasks of VCE system ($M=100$, $m=10$)

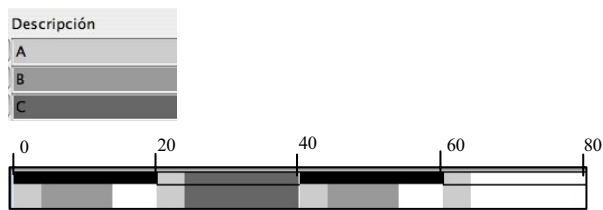


Figure 4: Cyclic scheduler of the group of tasks of Table 2: Set of tasks of a car control system.

Figure 4 shows the simulation of the cyclic scheduler given.

5 CONCLUSIONS AND FUTURE WORK

This paper deals with the introduction of CICLIC for the analysis, design, and development of cyclic scheduling on real-time systems based on a processor.

One of the main characteristics of CICLIC is the complementary use of analytical and simulation components during the design process. Firstly, CICLIC checks the completion of all requirements for task scheduling; then, it determines task scheduling and simulation. CICLIC is based on secondary cycle scheduler and on heuristic search algorithms to get a feasible scheduling.

Our future work includes expanding this tool in order to deal with group of tasks having precedence relationships and mutual exclusion. We also plan to introduce and evaluate new heuristic rules that may increase the effectiveness of this tool.

ACKNOWLEDGEMENTS

The research for this paper has been sponsored by the Ministry of Science and Technology, through reference project No. DPI20002/01621.

REFERENCES

- [1] T. P. Baker and A. Shaw. The Cyclic Executive Model and Ada *Real Time Systems* 1(1), 1989.
- [2] G. Buttazzo. *Hard Real-Time Computer Systems. Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
- [3] J. Blumenthal, F. Golasowski, J. Hildebrandt, and D. Timmermann. *Framework for validation and Analysis of Real Time Scheduling Algorithms and scheduler implementations*. University of Rostock, Technical report available from <http://yasa.e-technik.uni-rostock.de/>, 2003.
- [4] A. Burns N. Hayes and M.F. Richardson, "Generating Feasible Cyclic Schedules," *Control Eng. Practice*, vol. 3, no. 2, pp. 151-162, 1995.
- [5] The Cheddar project: A free real time scheduling analyzer, <http://beru.univ-brest.fr/~singhoff/cheddar/>
- [6] S. Devroey, J. Goossens, and C. Hernalsteen. A generic simulator of real-time scheduling algorithms, pp. 242–249. *Proceedings of the 29th Annual Simulation Symposium*, New Orleans, Louisiana, April 1996.
- [7] J. Flores Zamorano, "Planificación Estática de Procesos en Sistemas de Tiempo Real Crítico", *Tesis Ph. D.*, Univ. Politécnica de Madrid, 1995.
- [8] M. R. Garey and D. Johnson *Computers and Intractability. A Guide to the Theory of NP-Completeness* Freeman, 1979.
- [9] M. G. Harbour, J. G. García, J. P. Gutierrez, and J. D. Moyano. MAST: Modeling and Analysis Suite for Real Time Applications, pp. 125–134. *Proc. of the 13th Euromicro Conference on Real-Time Systems*, Netherlands, June 2001.
- [10] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real Time Analysis*. Kluwer Academic Publishers, 1994.
- [11] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.
- [12] J. Mäki-Turja, K. Hänninen, M. Nolin. Efficient Development of Real-Time Systems Using Hybrid Scheduling, *International conference on Embedded Systems and Applications (ESA)*, June, 2005
- [13] SEI. The Rate Monotonic Analysis. Technical report, *In the Software Technology Roadmap*. <http://www.sei.cmu.edu/str/descriptions/rma/body.html>, September 2003.
- [14] TimeSys. Using TimeWiz to Understand System Timing before you Build or Buy. *White paper*, http://www.timesys.com/index.cfm?bdy=home_dlibrary.cfm, 2002.
- [15] Tri-Pacific. *Rapid-RMA: The Art of Modeling Real-Time Systems*. <http://www.tripac.com/html/prod-fact-rrm.html>, 2003.
- [16] J. A. Stankovic Misconceptions about real-time programming *IEEE Computer*. Octubre 1988-
- [17] Volvo Construction Equipment. <http://www.volvoce.com>.
- [18] J. Yépez, P. Martí and J. M. Fuertes. "Control Loop Scheduling Paradigm in Distributed Control Systems." *29th Annual Conference of the IEEE Industrial Electronics Society (IECON03)*, Roanoke, USA, November, 2003.