Parallel Algorithms for the Two-Dimensional Cutting Stock Problem

Coromoto León Hernández **Gara Miranda Valladares**
Casiano Rodríguez León **Carlos Segura González**

Dpto. de Estadística, I.O. y Computación

Universidad de La Laguna

Outline

- Introduction
- Improvements to the Sequential Algorithm
- Parallel Algorithm
- Synchronization Service
- Computational Results
- Conclusions

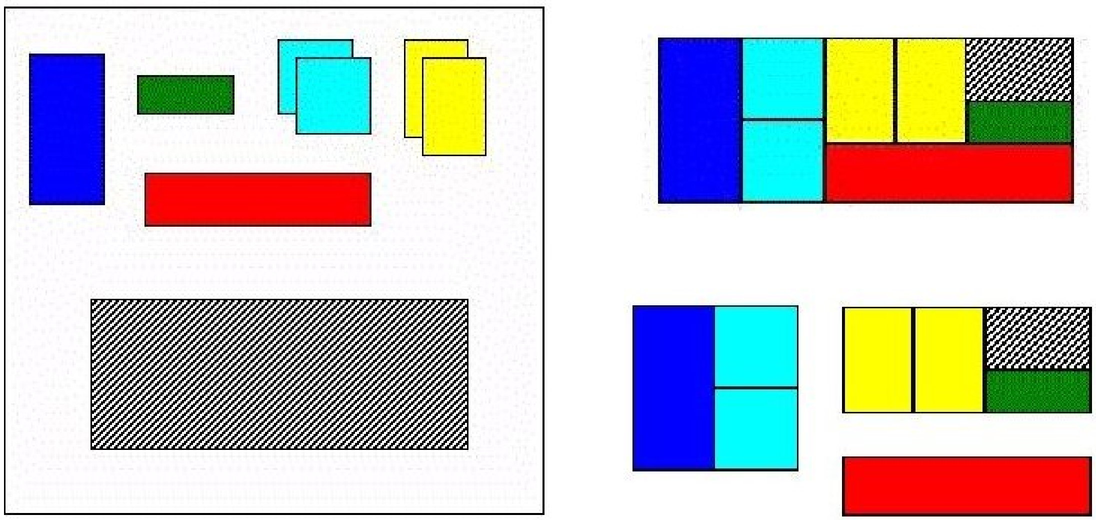
Cutting Stock Problem (CSP)

- CSP arise in many production industries.
- Large stock sheets (glass, textiles, paper, etc.) must be cut into smaller pieces.
- CSP can be classified attending to:
  - the number of dimensions (1D, 2D, 3D)
  - the number of available surfaces and patterns
  - the shape of the patterns (regular or irregular)
  - the orientation

Constrained Two-Dimensional Cutting Stock Problem

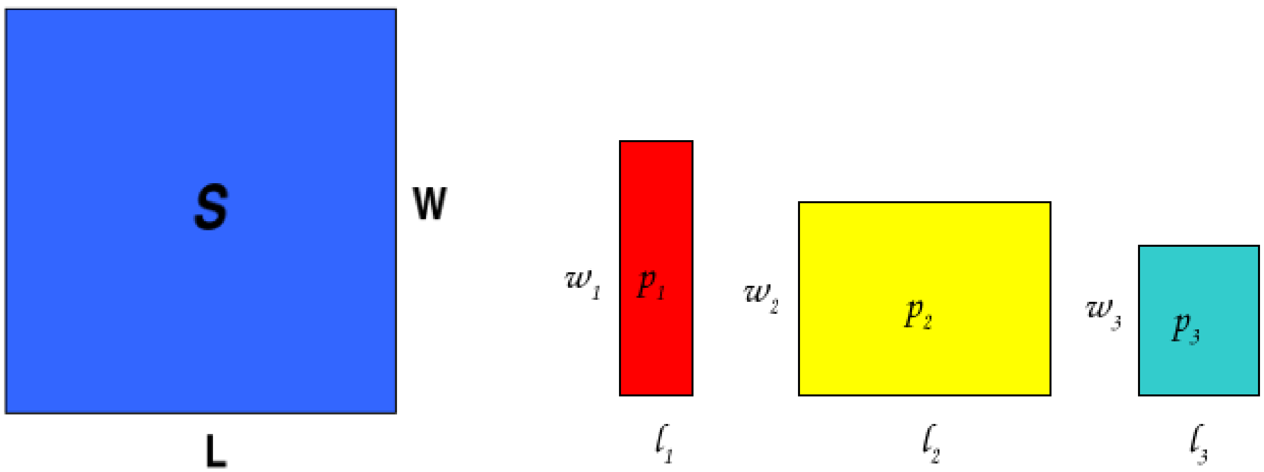
- The Constrained 2DCSP is one of the most interesting variants of CSP and targets the cutting of a large rectangle S of dimensions $L \times W$ in a set of smaller rectangles using orthogonal guillotine cuts.

- Any cut must run from one side of the rectangle to the other end and be parallel to the other two edges.



- The produced rectangles must belong to one of a given set of rectangle types $\mathcal{D} = \{T_1 \ldots T_n\}$ where the i-th type T_i has dimensions $l_i \times w_i$.

- Associated with each type T_i there is a profit p_i and a demand constraint b_i.



- The problem goal is to find a feasible cutting pattern with x_i pieces of type T_i maximizing the total profit:

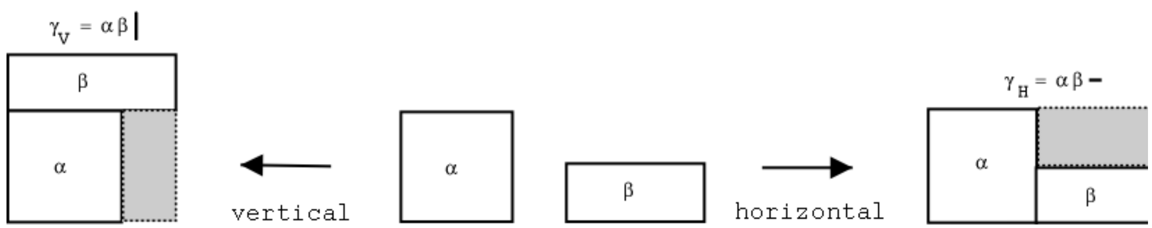
$$Maximize \sum_{i=1}^{n} x_i p_i \text{ subject to } x_i \leq b_i \text{ and } x_i \in \mathbb{N}$$

Solving the Constrained Two-Dimensional CSP

- Non-exact algorithms:
  - Heuristics
  - Evolutionary algorithms
- Exact algorithms:
  - Depth-first searches (*Christofides & Whitlock (1977)*)
  - Best-first searches (*Viswanathan & Bagchi (1993), Hifi (1997), Cung et al. (1997)*)
- Parallel approximations:
  - Parallel version of Wang's approximation (*Niklas et al. (1998)*)
  - Parallel version based on original Viswanathan & Bagchi algorithm and PPBB-LIB (*Tschöeke & Holthöfer (1995)*)

Viswanathan and Bagchi's Algorithm

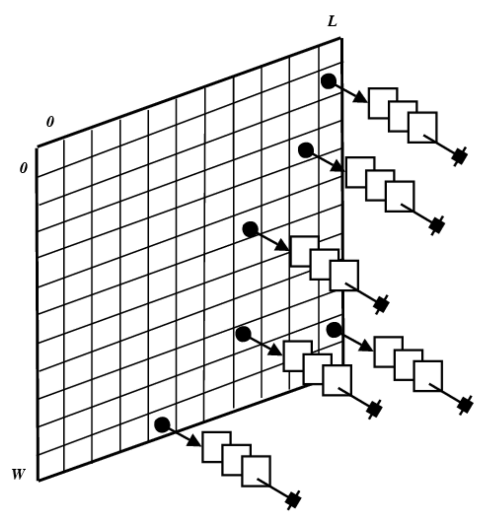
- The algorithm needs two lists of builds (subproblems):
  - OPEN stores all the generated builds that are still pending to be analysed.
  - CLIST stores the best builds that have been analysed.
- At each step, an element α with dimensions (α^l, α^w) is removed from OPEN and inserted into CLIST.
- This element is combined with the elements in CLIST in order to generate all the new horizontal $\gamma_H = (\alpha\beta-)$ and vertical $\gamma_V = (\alpha\beta|)$ builds (*Wang (1983)*).



- The element from OPEN to be selected must be the one with the highest *estimated total profit* (best-first search scheme).

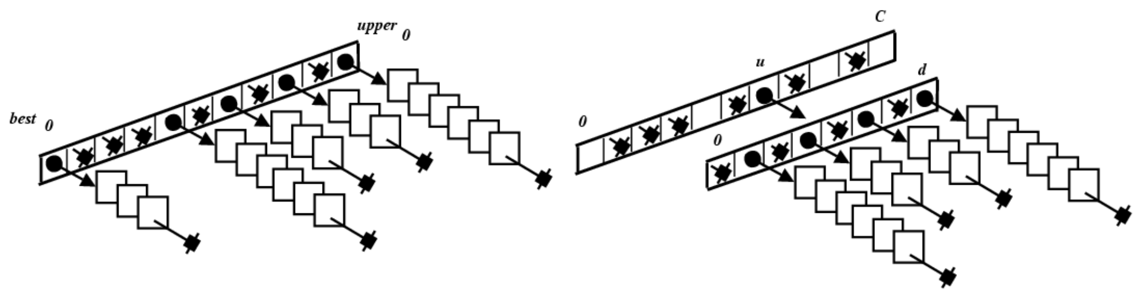
Modified Viswanathan and Bagchi's Algorithm *(Cung et al, 1997)*

- In VB original version the combination is achieved traversing the whole CLIST.
- The new data structure for CLIST alleviate the generation of non-feasible builds.



- Improvements of the lower and upper bounds.
- Detection of duplicated/dominated builds.

- New data structure to store *OPEN*:
  - Subproblems are sorted by the value of their upper bounds (best-first search).
  - Lower bounds keep ascending and the upper bounds descending (Branch-and-Bound).
  - When there is no space to afford storing the whole interval $[best_0, upper_0]$ the data structure becomes a tree-of-intervals.
  - Insertions can be done in constant time.
  - Full segments of memory can be freed any time the lower bound improves.



- Any feasible solution can be represented using postfix expressions.
- Shared memory parallelization of the subproblem generation loop.

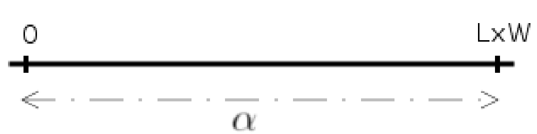
A New Upper Bound

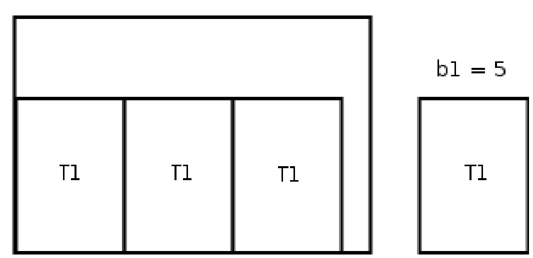
1. The following bounded knapsack problem is solved using dynamic programming:

$$V(\alpha) = \begin{cases} \max \sum_{i=1}^{n} c_i x_i & \\ \text{subject to} & \sum_{i=1}^{n} (l_i w_i) x_i \leq \alpha \\ \text{and} & x_i \leq min\{b_i, \lfloor \frac{L}{l_i} \rfloor \times \lfloor \frac{W}{w_i} \rfloor\}, x_i \in \mathbb{N} \end{cases}$$

for all $0 \leq \alpha \leq L \times W$

- Consider all the possible areas of the larger piece.
- Maximize the profit of the considered area.
- Constraints on the maximum number of pieces to use: dimensions and availability.





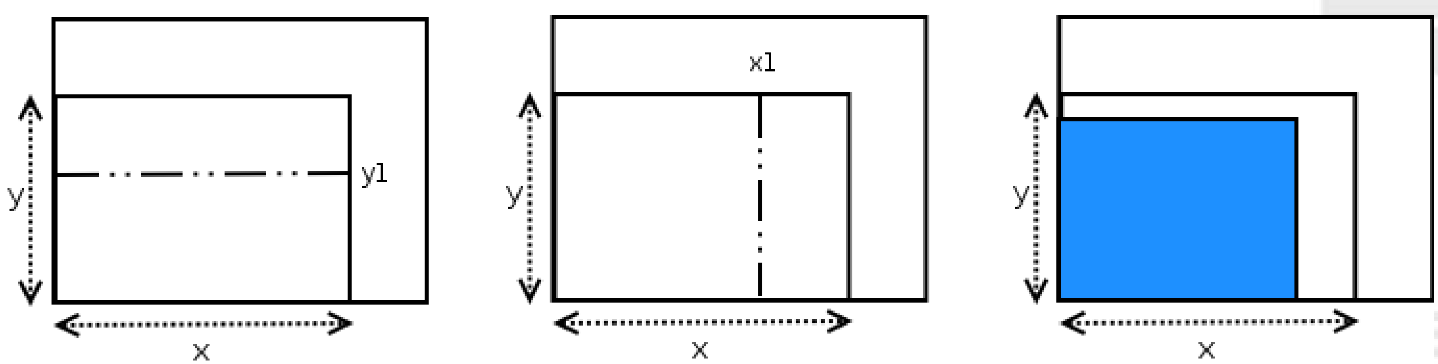
2. Then, $F_V(x, y)$ is computed for each rectangle using the equations:

$$\overline{F}(x,y) = \max \begin{cases} \max\{F_V(x,y_1) + F_V(x,y-y_1) & \text{such that } 0 < y_1 \le \lfloor \frac{y}{2} \rfloor\} \\ \max\{F_V(x_1,y) + F_V(x-x_1,y) & \text{such that } 0 < x_1 \le \lfloor \frac{x}{2} \rfloor\} \\ \max\{c_i & \text{such that } l_i \le x \text{ and } w_i \le y\} \end{cases}$$

where

$$F_V(x,y) = min\{\overline{F}(x,y), V(x \times y)\}$$

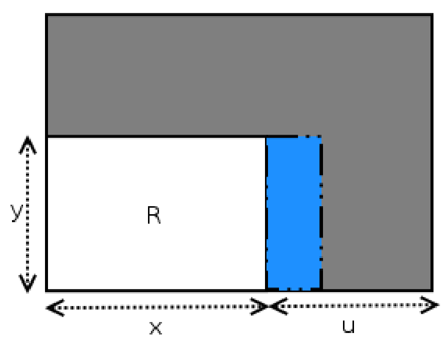
- Consider all the possible vertical and horizontal subdivisions of the surface (x, y).
- Consider the individual piece that maximize the profit of the surface (x, y).

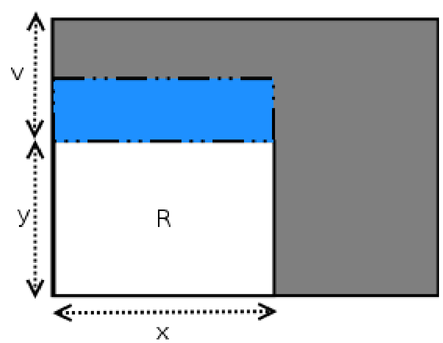


3. Finally, substituting the bound of Gilmore and Gomory by F_V in Viswanathan and Bagchi upper bound the new proposed upper bound is obtained:

$$U_V(x,y) = \max \begin{cases} \max\{U_V(x+u,y) + F_V(u,y) & \text{such that } 0 < u \leq L - x\} \\ \max\{U_V(x,y+v) + F_V(x,v) & \text{such that } 0 < v \leq W - y\} \end{cases}$$

- Enumerate all possible ways such a rectangle R of dimensions (x,y) is at the bottom-left corner of some guillotine cutting pattern.
- Two possibilities: horizontal or vertical construction.
- Profit of the additional considered build plus the profit of the remaining area.



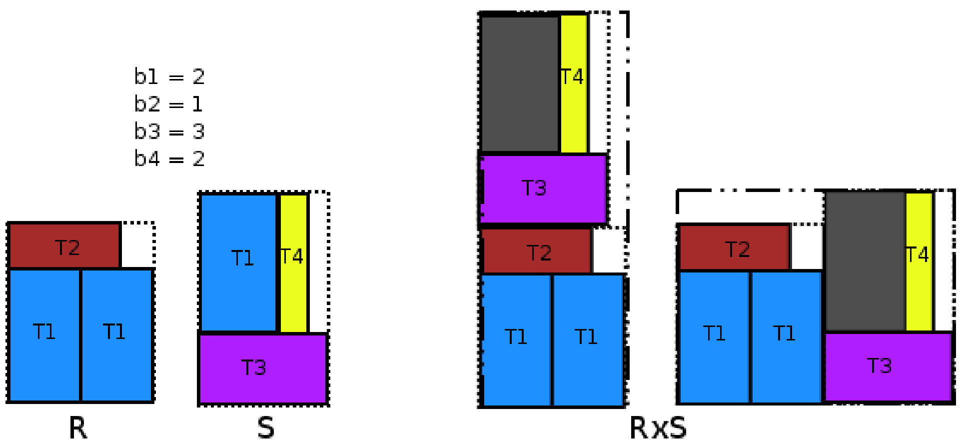


A New Lower Bound

- Mimics Gilmore and Gomory dynamic programming algorithm, but substituting unbounded vertical and horizontal combinations by feasible suboptimal ones.

- Let be $R = (r_i)_{i=1\ldots n}$ and $S = (s_i)_{i=1\ldots n}$ sets of feasible solutions using $r_i \leq b_i$ and $s_i \leq b_i$ rectangles of type T_i.

- The cross product $R \otimes S$ of R and S is defined as the set of feasible solutions built from R and S without violating the bounding requirements:
  - $R \otimes S$ uses $(\min\{r_i + s_i, b_i\})_{i=1\ldots n}$ rectangles of type T_i.

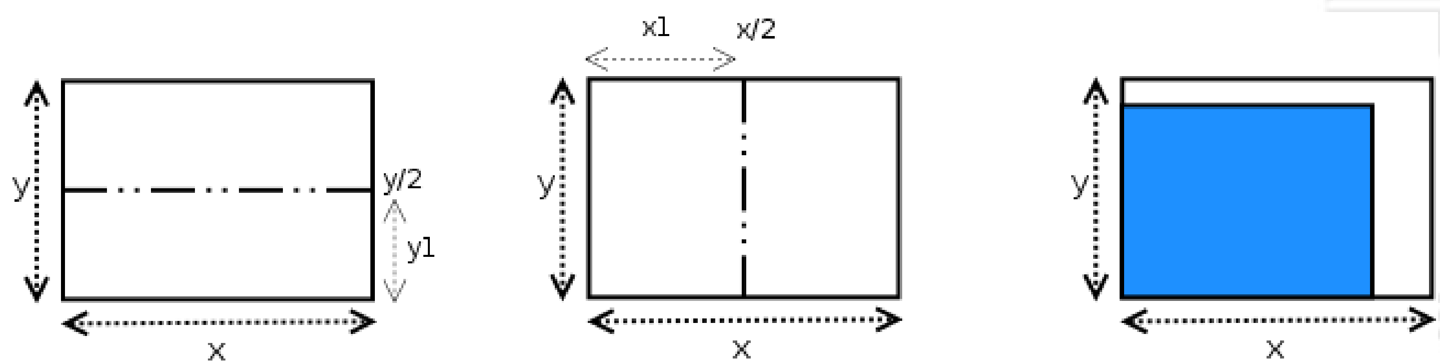


- The lower bound is given by the value $H(L, W)$ computed by:

$$H(x,y) = \max \begin{cases} \max\{g(S(x,y_1) \otimes S(x,y-y_1)) & \text{such that } 0 < y_1 \leq \lfloor \frac{y}{2} \rfloor\} \\ \max\{g(S(x_1,y) \otimes S(x-x_1,y)) & \text{such that } 0 < x_1 \leq \lfloor \frac{x}{2} \rfloor\} \\ \max\{c_i & \text{such that } l_i \leq x \text{ and } w_i \leq y\} \end{cases}$$

being $S(x, y)$ the build where the maximum is reached.

- Consider all the possible vertical and horizontal subdivisions of the surface (x, y).
- Consider the individual piece that maximize the profit of the surface (x, y).



General Operation

- The parallel algorithm is partially based on VB modified version.
- Every processor has its own local CLIST and OPEN:
  - CLIST is replicated and OPEN is distributed among the available processors.
- The initial builds are distributed among the processors.
- Each processor independently works as in the improved sequential scheme.
- Every certain periods of time, all processors have to do an exchange of computed subproblems in order to generate the complete set of feasible solutions.
  - Synchronization based on the number of search-loop iterations or number of computed/generated nodes.
  - Irregular cost associated to each loop iteration or computed/generated node.
- The stop condition is reached when all the OPEN lists are empty.

Sequential Case	
OPEN $= \{a,\ b,\ c\}$	CLIST $= \{\ \}$
OPEN $= \{aa-,\ b,\ aa\vert,\ c\}$	CLIST $= \{a\}$
OPEN $= \{b,\ aa\vert,\ c\}$	CLIST $= \{a,\ aa-\}$
OPEN $= \{ba\vert,\ bb\vert,\ aa\vert,\ ba-,\ c\}$	CLIST $= \{a,\ aa-,\ b\}$

Parallel Case		
Processor 1	OPEN $= \{a\}$	CLIST $= \{\}$
Processor 2	OPEN $= \{b\}$	CLIST $= \{\}$
Processor 3	OPEN $= \{c\}$	CLIST $= \{\}$
Processor 1	OPEN $= \{aa-,\ aa\vert\}$	CLIST $= \{a\}$
Processor 2	OPEN $= \{bb\vert\}$	CLIST $= \{b\}$
Processor 3	OPEN $= \{cc-,\ cc\vert\}$	CLIST $= \{c\}$
Processor 1	OPEN $= \{aa-,\ aa\vert\} + \{ab$ builds$\}$	CLIST $= \{a,\ b,\ c\}$
Processor 2	OPEN $= \{bb\vert\} + \{bc$ builds$\}$	CLIST $= \{b,\ a,\ c\}$
Processor 3	OPEN $= \{cc-,\ cc\vert\} + \{ca$ builds$\}$	CLIST $= \{c,\ a,\ b\}$

Communication Scheme

- It has been implemented using a *synchronization service*.
- The synchronization subroutine is called when:
  - a processor has no pending work
  - an active alarm of the synchronization service goes off
- The information given by each processor consists of:
  - best solution value
  - OPEN list size
  - set of builds analyzed since the last synchronization step
- Information to be updated by each processor:
  - Elements computed by other processors must be inserted into the local CLIST
  - Combinations of computed elements are uniformly distributed among processors
  - Local best solution is updated with the best solution found by any of the processors

Load Balancing Scheme

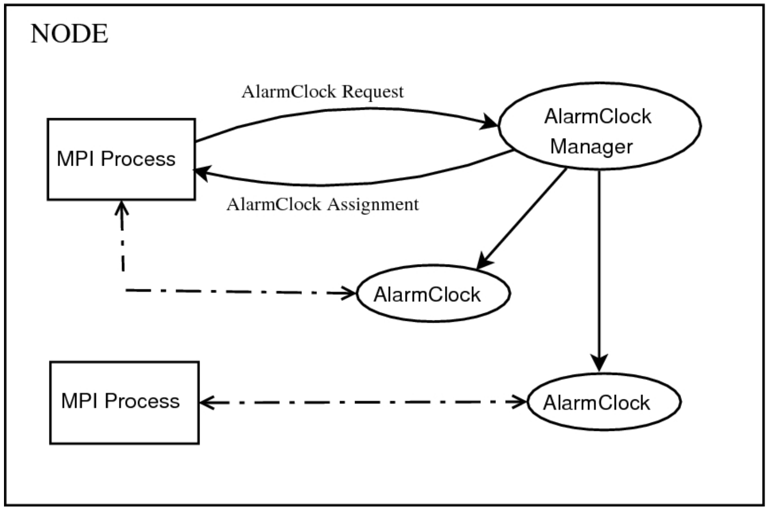
- Requires three configuration parameters:
  - *MinBalThreshold*, *MaxBalThreshold*, *MaxBalanceLength*.

- The method is executed after the computation of the pending combinations.

- Operation:

(a) Sort the set of processors attending to their OPEN size.
(b) Match the processor with largest OPEN list with the processor with the smallest one, the second largest one with the second smallest and so on.
(c) Partners will make an exchange of elements if the one with larger OPEN has more than *MaxBalThreshold* elements and the other has less than *MinBalThreshold*.
(d) The number of elements to be exchanged is proportional to the difference of the two OPEN sizes, but it can never be greater than *MaxBalanceLength*.

- All synchronizations in the model are done through time alarms (*alarm clocks*).
- Service independent of the particular algorithm and the MPI implementation.
- Using the service:
  - By using a daemon, an *alarm clock manager* is created on each node.
  - For each received request, the service manager creates a new *alarm clock process* that will communicate to the corresponding requester.



  - Algorithmic processes can activate/cancell alarm clocks.
  - When an alarm goes off, the corresponding process is notified.

Description of the Experiments

- For the computational study, we have selected some CSP instances from the ones available in the related literature.
- Tests have been run on a cluster of 8 HP nodes, each one consisting of two Intel(R) Xeon(TM) at 3.20GHz.
- The interconnection network is an Infiniband 4X SDR.
- The compiler and MPI implementation used were *gcc 3.3* and MVAPICH *0.9.7*.
- Sequential tests:
  - Comparison of the original lower bound and the new one.
  - Comparison of the original upper bound and the new one.
- Parallel tests:
  - Executions with 1, 2, 4, 8, 16 processors.
  - Comparison of execution times and number of computed nodes.

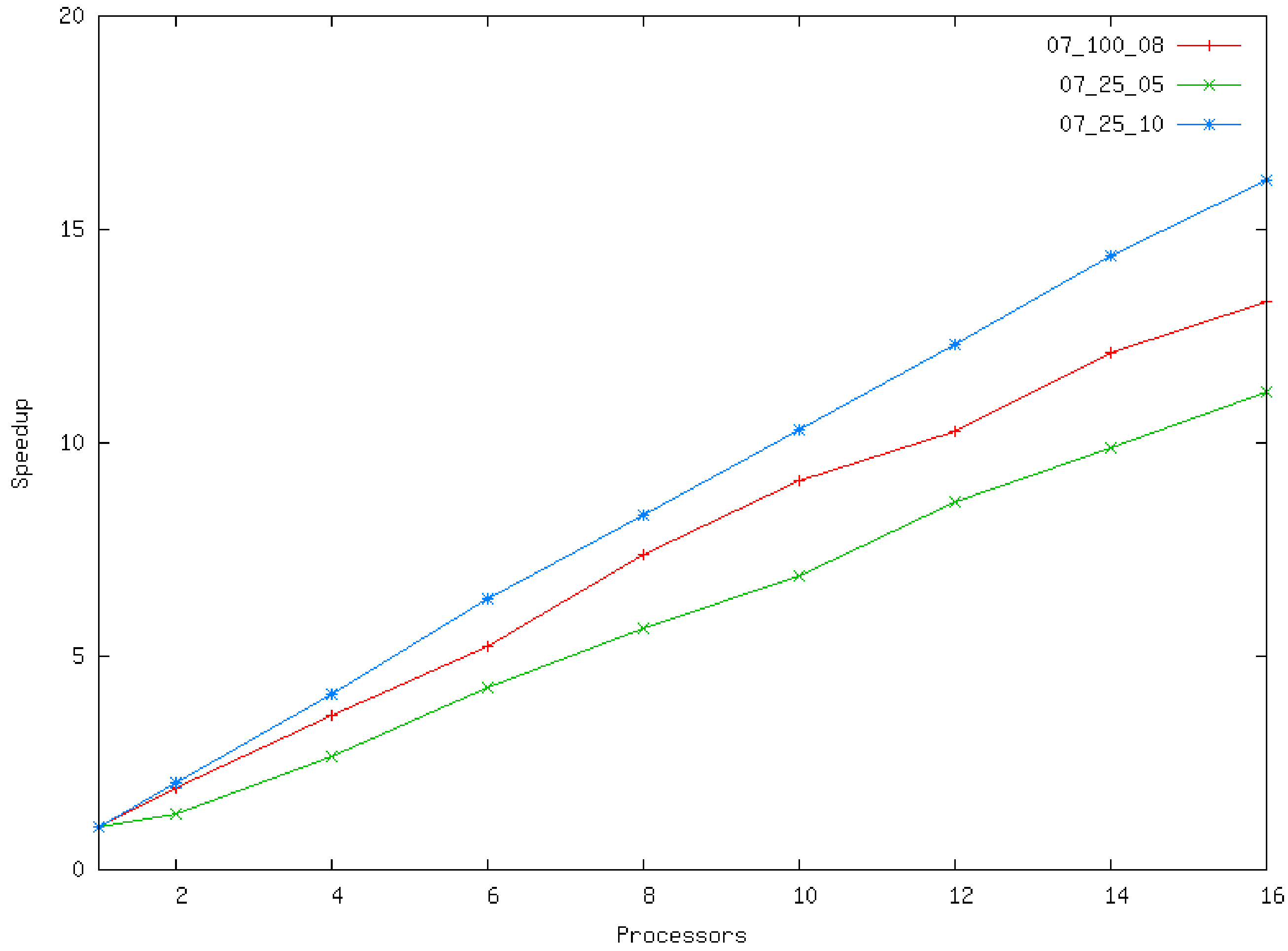
Lower and Upper Bounds Results

Problem	Solution	Lower Bound		Upper Bound					
				V			U_V		
	Value	Value	Time	Init	Search	Nodes	Init	Search	Nodes
25_03	21693	21662	0.442	0.0309	2835.07	179360	0.0312	2308.78	157277
25_05	21693	21662	0.436	0.0311	2892.23	183890	0.0301	2304.78	160932
25_06	21915	21915	0.449	0.0316	35.55	13713	0.0325	20.83	10310
25_08	21915	21915	0.445	0.0318	205.64	33727	0.0284	129.03	25764
25_09	21672	21548	0.499	0.0310	37.31	17074	0.0295	25.49	13882
25_10	21915	21915	0.510	0.0318	1353.89	86920	0.0327	1107.18	73039
50_01	22154	22092	0.725	0.1056	2132.23	126854	0.0454	1551.23	102662
50_03	22102	22089	0.793	0.0428	4583.44	189277	0.0450	3046.63	148964
50_05	22102	22089	0.782	0.0454	4637.68	189920	0.0451	3027.79	149449
50_09	22088	22088	0.795	0.0457	234.42	38777	0.0428	155.35	29124
100_08	22443	22443	1.218	0.0769	110.17	25691	0.0760	92.91	22644
100_09	22397	22377	1.278	0.0756	75.59	20086	0.0755	61.84	17708

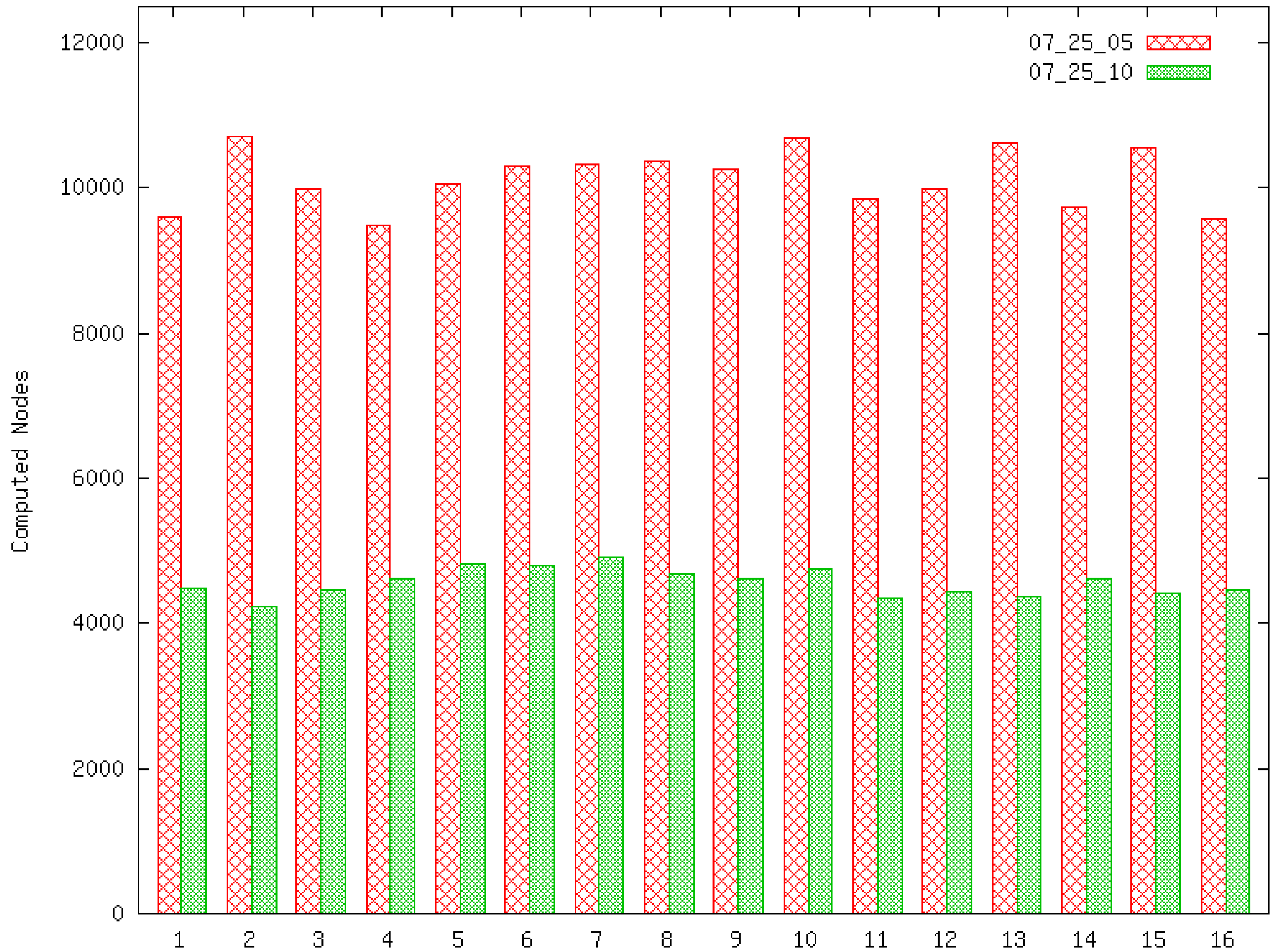
Parallel Algorithm Results

PROBLEM	PROCESSORS 1 Time	1 Nodes	2 Time	2 Nodes	4 Time	4 Nodes	8 Time	8 Nodes	16 Time	16 Nodes	16 Sp.
25_03	2922.94	157277	1665.26	161200	770.47	157281	384.05	159424	197.82	157603	11.67
25_05	3068.19	160932	1738.02	168941	863.23	168867	408.39	165323	206.10	162029	11.18
25_06	23.82	10310	11.51	10310	6.36	10310	3.01	10310	1.57	10310	13.26
25_08	129.02	25764	61.38	25764	29.98	25764	15.52	25764	8.33	25764	15.48
25_09	29.44	13882	13.69	14257	7.02	13916	3.57	13916	2.09	14150	12.44
25_10	1140.41	73039	539.89	73039	266.96	73039	132.32	73039	67.94	73039	16.16
50_01	1651.51	102662	963.07	102662	598.67	116575	240.93	103545	123.72	102965	12.53
50_03	4214.54	148964	2084.77	148964	1057.70	151362	512.12	150644	258.51	149039	11.78
50_05	4235.27	149449	2141.41	149449	1077.47	153813	512.43	150937	260.03	149450	11.64
50_09	161.38	29124	77.65	29124	40.34	29124	19.45	29124	10.34	29124	14.94
100_08	98.96	22644	48.74	22644	25.83	22644	12.60	22644	6.98	22644	13.31
100_09	60.05	17708	38.29	19987	18.74	18509	10.59	20584	4.77	18100	12.58

Computational Results - SpeedUp



Computational Results - Load Balancing



- Computational results prove the quality of the new lower and upper bound.
- All the approaches to parallelize VB strive against the highly irregular computation structure of the algorithm and its intrinsically sequential nature.
- A new parallel distributed and synchronous algorithm has been designed from the basis of the inherently sequential VB algorithm.
- Parallel results demonstrate the almost linear speedups and verify the high scalability of the implementation.
- A totally application-independent synchronization service has been developed.
- The service provides an easy way of introducing periodic synchronizations in the user programs.
- The synchronization service has been decisive for the well operation of the parallel scheme and for the right behaviour of the load balancing model.

- Improvements of the load balancing scheme:
  - Instead of considering only the size of the lists, it would be fairly to introduce some method to approximately calculate the work associated to each of the subproblems in OPEN.
- Improvements of the synchronization scheme:
  - At the initial and latest stages of the search, many of the alarms are cancelled because processors do not have enough work.
  - It would be interesting to have an automatic and dynamic way of fixing the time between synchronizations while the search process is progressing.

Thank you for your attention!

Questions?

This work has been supported by the EC (FEDER) and by the Spanish Ministry of Education inside the 'Plan Nacional de I+D+I' with contract number TIN2005-08818-C04-04.

The work of G. Miranda has been developed under the grant FPU-AP2004-2290.