

Design and use of the CPAN Branch & Bound for the solution of the Travelling Salesman Problem (TSP)

Mario Rossainz López
Facultad de Ciencias de la
Computación
Benemérita Universidad Autónoma
de Puebla
C/ San Claudio y 14 Sur, San
Manuel – 72000 Puebla, México
E-mail: mariorl@siu.buap.mx

Manuel I. Capel Tuñón
Dpt. Lenguajes y Sistemas
Informáticos.
E.T.S. Ingeniería Informática
Universidad de Granada
C/ Periodista Daniel Saucedo Aranda
s/n – 18071 Granada, Spain
E-mail: mcapel@ugr.es

KEYWORDS

Parallel Objects, Structured Parallel Programming, High Performance Computing, Object Oriented Programming, Concurrent Programming.

ABSTRACT

This article presents the design of a High Level Parallel Composition or CPAN (according to its Spanish acronym) that implements a parallelization of the algorithmic design technique named Branch & Bound and uses it to solve the Travelling Salesman Problem (TSP), within a methodological infrastructure made up of an environment of Parallel Objects, an approach to Structured Parallel Programming and the Object-Orientation paradigm. A CPAN is defined as the composition of a set of parallel objects of three types: one object manager, the stages and the Collector objects. By following this idea, the Branch & Bound design technique implemented as an algorithmic parallel pattern of communication among processes and based on the model of the CPAN is shown. Thus, in this work, the CPAN Branch & Bound is added as a new pattern to the library of classes already proposed in [9], which was initially constituted by the CPANs Farm, Pipe and TreeDV that represent, respectively, the patterns of communication Farm, Pipeline and Binary Tree, the latter one implementing the design technique known as Divide and Conquer. As the programming environment used to derive the proposed CPANs, we use C++ and the POSIX standard for thread programming.

INTRODUCTION

The present investigation centres its attention on the Methods of Structured Parallel Programming, proposing a new implementation, carried out with C++ and the POSIX Threads Library, as a CPAN (Corradi and Zambonelli 1995; Danelutto, and Orlando) of the algorithmic design technique known

as Branch & Bound (BB). CPANs are Structured Parallel constructs based on the Object-Orientation paradigm useful to solve problems of high computational complexity by parallelizing their algorithms using a class of concurrent active objects. In this work the library of classes that we propose in (Rossainz and Capel 2004) is complemented with the design and implementation of the CPAN Branch & Bound, which is intended to provide the programmer with an additional communication and interaction pattern among processes in parallel applications, which allows him to solve optimization problems, such as the Travelling Salesman Problem discussed here, which is an optimization problem with NP-Complete complexity.

BRANCH AND BOUND METHOD

Branch-and-bound (BB) makes a partition of the solution space of a given optimization problem. The entire space is represented by the corresponding BB *expansion tree*, whose root is associated to the initially unsolved problem; the children at each node represent the subspaces obtained by *branching*, i.e. subdividing, the solution space represented by the parent node; and the leaves of the tree represent nodes that cannot be subdivided any further, thus providing a final value of the cost function associated to a possible solution of the problem. BB carries out a partial enumeration, i.e. a non-exhaustive search, over the nodes of the expansion tree until an optimal solution to the initial problem is found or the set of *live nodes*, i.e. those that still have the possibility of being branched, becomes exhausted.

There are different possibilities to generate nodes and follow a route to a solution during the algorithm execution, known as the branching strategies of the BB algorithm, such as the ones given by the following search methods: *First in depth* (strategy LIFO), *First in width* (strategy FIFO) and *First best node*. The latter uses cost functions calculation to select the node that in principle seems to be more promising to explore, i.e. to further expand in order

to find better solutions from it (strategy HEAP, using minimum cost or LC). In addition to these strategies, BB fixes bounds to the values of the suboptimal solution found at a certain point of the algorithm execution in order to prune those branches below a node that cannot lead to the optimal solution. A bound of the possible value of those reachable solutions is calculated in each node from the information contained in it. If the bound shows that any one of these solutions is necessarily worse than the best solution found up to that point, then there is no need for the algorithm to continue exploring on that branch and, therefore, prunes it off.

The Algorithm

Three stages are carried out in BB algorithms:

1. **Selection:** A node belonging to the set of live nodes is extracted. The selection directly depends on the strategy search which was decided for use in the algorithm.
2. **Branch:** the node selected in the previous step is subdivided in its children nodes by following a ramification scheme to form the expansion tree. Each child node receives from its father node enough information to enable it to search a suboptimal solution.
3. **Bound:** Some of the nodes created in the previous stage are deleted, i.e. those whose partial cost, which is given by the cost function associated to this BB algorithm instance, is greater than the best minimum bound calculated up to that point.

The contribution of this algorithm is that it offers a way to perform the greatest possible reduction of the space search and, therefore, obtains a decrease in the exploration complexity of the expansion tree which contains the optimal solution being searched. The nodes that have not yet been pruned are included in the live node list, and thereby, the selection process begins again until the algorithm finalizes. The general structure of the algorithms that implement the BB technique is based on three main modules:

1. The module that contains the scheme of general operation of the technique.
2. The module that represents the structure of data where the generated nodes are stored.
3. The module that describes the structures of data that conform the nodes.

The first module is the only one that remains without modification, independently of the problem to solve with the BB algorithm and it is valid for all the algorithms that follow the technique. The pseudo-code implementing it is as follows:

```

CLASS CONCRETE EsquemaBB
{
    Estructura e;
    Nodo n;
    Nodo[] hijos;
    int numhijos,i,j;

    PUBLIC Nodo B&B()
    {
        e= Estructura CREATE();
        n= nodoInicial();
        e.inserta(n,n.h());
        WHILE (!E.esVacía())
        {
            n=e.extrae();
            numhijos=n.expandir(VAR hijos);
            eliminar(n);
            n.poner_cota_sup(numhijos,hijos);
            FOR i=(0,numhijos)
            {
                IF (n.aceptable(hijos[i]))
                {
                    IF (n.esSolucion(hijos[i]))
                    {
                        FOR j=(0,numhijos)
                            IF (i!=j)
                                DELETE hijos[j];
                        e.clear();
                        RETURN hijos[i];
                    }
                    ELSE
                        e.inserta(hijos[i],
                                n.h(hijos[i]));
                }
                ELSE
                    DELETE hijos[i];
            }
        }
    }
}

```

The definition of the abstract data type that represents the structure e where the nodes are stored corresponds to the ADT HEAP, because the strategy used to search a node containing a solution is that of selecting the minimum cost (LC) one among the contained in the HEAP. The definition of the class that represents the type *Node* used in the previous pseudo-code is composed of the following functions:

- **expandir():** It is the function that creates the children nodes out of a given node and returns the number of children to where the function is called. This function is the one that implements the process of node ramification of the algorithm
- **aceptable():** Function that carries out the pruning of unpromising nodes and, when it obtains a live node, decides whether to continue exploring or to reject it.
- **esSolucion():** It is a function that decides whether its parameter node is a leaf of the tree, that is to say, a possible solution to the original problem.
- **h():** This function in its two versions, i.e. it can be overloaded, is the one that implements the cost

function for the search strategy LC and its value is used as a priority position value when storing the nodes in the HEAP structure.

- **poner_cota_sup():** It allows for the upper bound of the problem to be established. The function that carries out node pruning uses this datum to prune those nodes whose cost value is greater than the currently obtained bound of the optimal solution.

The CPANS can provide the parallel algorithms necessary to solve problems, such as the one of the TSP, using the BB technique. When solving this kind of problems, in addition to the solution, reasonable run times of the whole computation can be obtained with the CPAN model. Since the complexity in NP-complete problems as in the TSP one is intrinsic, parallelization is the only way to obtain a solution in the practice. Each object node of the expansion tree therefore needs to be independent that is to say, it must contain all the necessary information to be an active object, (i.e. to have the capability of execution in itself) which makes it possible for the processes of branch and bound to perform the reconstruction of the solution found up to that moment.

THE TRAVELLING SALESMAN PROBLEM (TSP)

The Travelling Salesman Problem could be represented by a directed graph consisting of a set of vertices (cities) and labelled arcs (distances between cities). One optimized solution of the TSP is a path in which all the vertices have been visited exactly once with minimal cost (Capel and Palma 1992). Formally, the problem can be enunciated as follows: given a connected and weighted graph g and given one of its vertices v_0 as the initial one, we must find the Hamiltonian cycle of minimum cost that begins and finishes in v_0 (Guerrequeta and Vallecillo).

Solution of the TSP with the Branch & Bound technique

The problem is solved by a BB algorithm, which dynamically builds a search tree, whose root is the initial problem and its answer nodes are complete tours (Capel and Palma 1992) around all the cities represented by the nodes of the graph. Numerous strategies of Branch & Bound exist that solve the problem of the TSP. The first of the three strategies described in (Horowitz 1978) is used in this work where the following important elements are defined, as well:

- **LC(P) - Cost of a node P:** Is the distance of a complete graph's tour after P gets included in it, if P is a solution node, or, otherwise, is the cost of a

solution of minimal cost in the sub-tree whose root is P .

- **cota - Upper bound:** The length of the shortest complete tour found up until that moment. This value must be a global variable in the program and is used to prune the search tree of nodes.
- **h(P) - lower bound** of the cost of a solution for a problem or sub-problem P . If P is a solution node $LC(P) = h(P)$.
- **s[] - vector solution:** It is a vector that indicates the order in which the vertices must be visited to reach the optimal solution. Each element of the vector contains a number between 1 and N , being N the number of vertices of the graph that defines the problem. The vector cannot contain repeated elements.
- **M[][] - Matrix of adjacency:** It is the representation of the graph where the vertices are the indices of the matrix and the contents of the matrix elements given are the arcs between two vertices. The matrix of adjacency is not necessarily symmetrical, although, it is so with respect to its nonnegative elements.

Computing the lower bound of a node of the search tree is carried out by obtaining the *reduced cost matrix* for each node (Capel and Palma 1992). A row (or column) of a matrix is reduced if it contains an element zero at least, and the rest of the elements are nonnegative. We say that a matrix is reduced if and only if all its rows and columns are reduced (Guerrequeta and Vallecillo). With respect to the interpretation of the cost of a node, this is obtained by adding the amounts t_i in which the rows or columns of the matrix of adjacency of the graph are reduced when the process of obtaining the reduced matrix is carried out. This amount which is removed when reducing a matrix is a lower bound of the total cost of any possible tour traversing the graph nodes. This is exactly what it is used as, the cost function LC to prune nodes p of the expansion tree. Therefore, a *reduced matrix* and an *accumulated cost* are associated to each node and if we suppose that $M'[][]$ is the reduced matrix associated to the node p and p' is a children node of p which is obtained by including the arc $\{i,j\}$ in the partially built tour $s[]$, then:

- If p' is a leaf node of the tree, i.e. a possible solution, its cost is the accumulated cost of p plus $M[i,j] + M[j,1]$. The latter term is the one that completes the tour. The amount obtained is the cost of such a tour.
- If p' is not a leaf, its reduced matrix M obtained from matrix M' assigned to this node p' will have as cost the cost of p added to the cost of the reduction of M' added to the value of $M[i,j]$.

DEFINITION OF CPAN

The basic idea is implementing different types of parallel patterns of communication between the processes of an application and implementing distributed/parallel algorithms as classes, by following the Object-Oriented paradigm. The execution of a method of the objects that constitute a CPAN can be carried out through a message sent to an instantiated class, which acknowledges it as a service petition.

A CPAN comes from the composition of a set of objects of three types (see Figure 1):

An object manager representing the CPAN itself and makes of it an encapsulated abstraction that hides its internal structure. The manager controls the references of a set of objects (a denominated object Collector and several denominated Stage objects) that represent the components of the CPAN and whose execution is carried out in parallel and should be coordinated by the manager itself.

The Stage objects are objects of specific purpose responsible for encapsulating a client-server type interface between the manager and the object slaves (objects that are not actively participative in the composition of the CPAN, but rather, are considered external entities that contain the sequential algorithm that constitutes the solution of a given problem) as well as providing the necessary connection among them to implement the communication semantic pattern that seeks to be defined. In other words, each stage should act in parallel as a node of the graph that represents the communication pattern and should be capable of executing its methods as an active object. A stage can be directly connected to the manager and/or to another component stage depending on the pattern peculiar to the CPAN being implemented.

And an object Collector which is an object in charge of storing in parallel the results that it receives from the stage objects that are connected to it. That is to say, during the service of a petition, the control flow within the stages of a CPAN depends on the implemented communication pattern. When the composition concludes its execution, the result does not return to the manager directly, but rather to an instance of the class Collector which takes charge of storing these results and of sending them to the manager, which will then send them to the exterior, as soon as they arrive to it, without the need of waiting for all the results to be obtained.

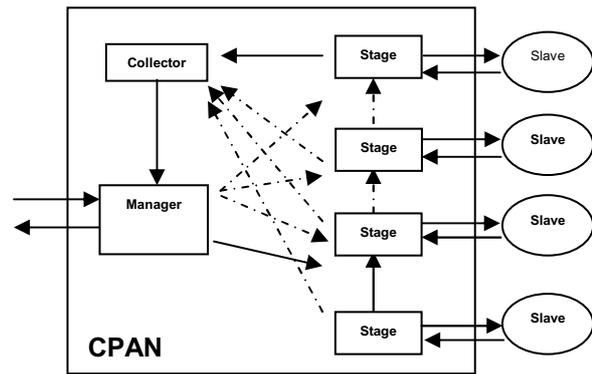


Figure 1. Internal Structure of a CPAN (Rossainz and Capel 2004).

The objects manager, collector and stages are included within the definition of Parallel Object (PO) (Corradi and Zambonelli 1995). The Parallel Objects are active objects, that is to say, objects that have execution and communication capabilities in themselves in three ways: the synchronous, the asynchronous and the asynchronous future way (Rossainz and Capel 2004).

In addition, for each one of these, synchronization mechanisms have been implemented when parallel petitions of service take place in a CPAN (*MaxPar*, *Mutex* and *Sync*) (Rossainz and Capel 2004).

The parallel patterns worked in (Rossainz and Capel 2004) have been the pipeline, the farm and the treeDV, to be a significant set of reusable patterns in multiple applications and algorithms. These patterns have been implemented on the basis of the model of the CPAN, constituting a library of High Level Parallel Compositions, formed by *the Cpan Farm*, *the Cpan Pipe* and *the Cpan TreeDV* Making use of the technique of the reusability of code and demonstrating the utility of the library proposed in (Rossainz and Capel 2004), the use of the *Cpan Farm* for the design of the technique of Branch & Bound like a CPAN becomes of interest. *The Cpan farm* is composed of a set of worker processes and a controller. The workers are executed in parallel until reaching a common objective. The controller is the one in charge of distributing work and of controlling the progress of the global calculation. See Figure 2.

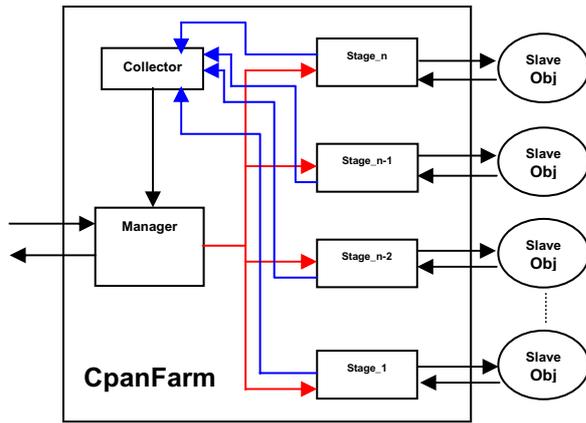


Figure 2. The Cpan of a Farm communication pattern (Rossainz and Capel 2004)

PARALELIZATION OF BRANCH & BOUND TECHNIQUE

The ramification is separated from the bounding of nodes on the expansion tree when the BB algorithm executes. These two structures were implemented using the *Cpan Farm* of the library proposed in (Rossainz and Capel 2004), so that the ramification and the distribution of work to the processes were carried out by using *the scheme of the Farm*. As figure 3 shows, the expansion tree, for a given instance of the BB algorithm, is obtained by iteratively subdividing the stage objects according to the farm pattern until a stage representing a leaf-node of the expansion tree is found, i.e., one stage in charge of solving a sub-problem that cannot be additionally subdivided.

On the other hand, the pruning is carried out implicitly within a *farm* construction by using a *scheme totally connected* between all the processes. It can communicate a sub-optimal bound found in a process to all the processes that are branching to avoid ramifications of useless sub-problems, i.e., those that do not lead to improving the best upper bound obtained up to that moment.

The *Cpan Branch & Bound* is composed of a set of *Cpans Farm* that represent worker processes and a controller, therefore, forming a new type of Farm, the *Farm Branch & Bound* or *FarmBB* that will be included in the library of CPANS. The *Cpan Farm* workers are executed in parallel forming the expansion tree of nodes given by this technique. The process controller of the initial *Cpan Farm* represents the root of the expansion tree that is in charge of distributing the work and of controlling the progress of the global calculation given to the collector of the *FarmBB* which sends the result to the process controller of the *Cpan FarmBB*, which then shows it to the user. See Figure 3.

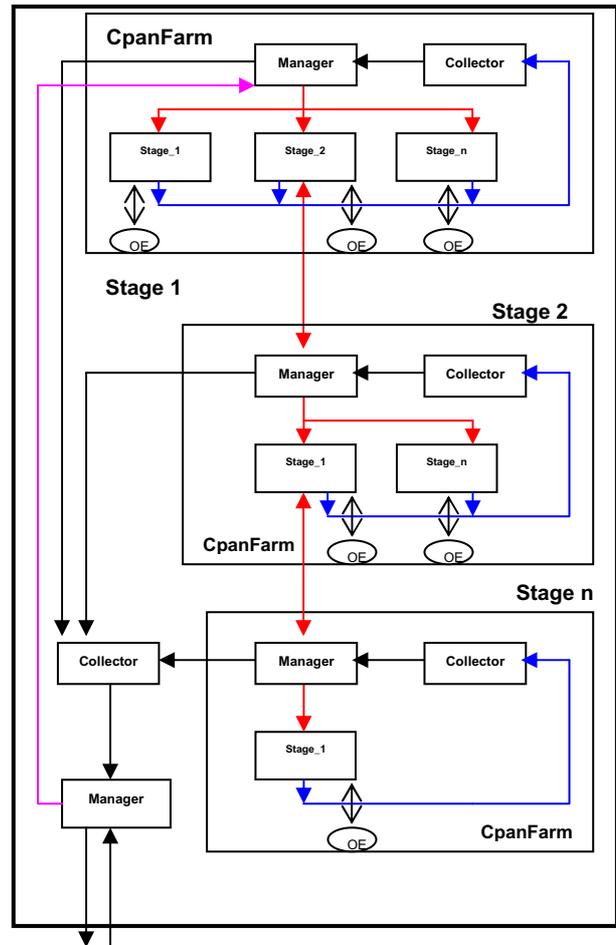


Figure 3. The Cpan Branch & Bound

RESULTS

The search strategy which was used in the implementation and test of the distributed TSP was: The first best search strategy that uses the calculation of cost functions for each live node to select the node that, in principle, seems the most promising to analyze (strategy HEAP, using minimum cost or LC). The graph taken from reference (GoodMan and Hedetniemi 1977) is an example of execution of the application considered. The graph represents a network of 5 cities whose matrix of costs is as follows:

	0	1	2	3	4
0	∞	25	40	31	27
1	5	∞	17	30	25
2	19	15	∞	6	1
3	9	50	24	∞	6
4	22	8	7	10	∞

The solution found was the path 0, 3, 4, 2, 1, 0, with a cost of 64, which is the same result found in the example in (GoodMan and Hedetniemi 1977).

ANALYSIS OF SPEEDUP

The analysis of Speedup of the CPANS B&B that appears in table 1 and Figure 4 was carried out in a Parallel System Origin 2000 Silicon Graphics (of 64 processors) available in the European Center for Parallelism of Barcelona CEPBA.

	CPU Seq.	CPU 2	CPU 4	CPU 8	CPU 16	Cpu 32
Run time	35.42 Seg.	21.88 Seg.	14.21 Seg.	11.34 Seg.	10.27 Seg.	9.10 Seg.
Time CPU	27.10 Seg.	23.25 Seg.	21.17 Seg.	19.19 Seg.	22.69 Seg.	22.18 Seg.
CPI	1.321	0.952	0.943	0.928	0.924	0.914
Speed Up	1.00	1.62	2.49	3.12	3.45	3.89
Amdal h	1.00	1.68	2.55	3.43	4.16	4.64

Table 1. Execution of Parallel CpanB&B in 2, 4, 8, 16 and 32 processors with N=50 cities

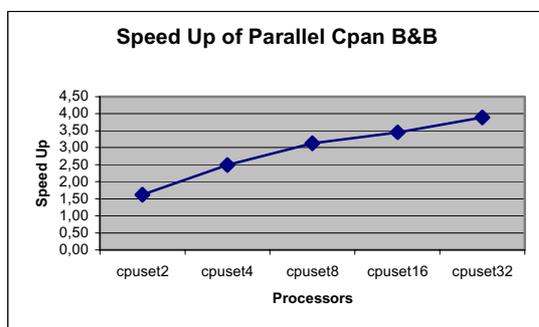


Figure 4. Speed Up of Parallel Cpan B&B

CONCLUSIONS

1. The technique of Branch & Bound as a High Level Parallel Composition or CPAN has been implemented.
2. The utility of the library of CPANS proposed in (Rossainz and Capel 2004) which serves to make compositions of CPANS and to define new CPANS models as in the *Cpan Branch & Bound* has been demonstrated.
3. With the model of the *Cpan Branch & Bound* we have been able to offer an optimal solution of a TSP NP-Complete problem.
4. The CPANS Pipe, Farm, TreeDV and Farm-Branch-&-Bound constitute the library of classes of the Cpan.

REFERENCES

- Brassard G.; Bratley P. 1998. "Fundamentos de Algoritmia". Prentice may. España. ISBN: 84-89660-00-X
- Capel M.I.; Palma A. 1992. "A Programming tool for Distributed Implementation of Branch-and-Bound Algorithms". Parallel Computing and Transputer Applications. IOS Press/CIMNE. Barcelona.

- Ceballos F.J. 2003. "Programación Orientada a Objetos con C++". Editorial RA-MA. España. ISBN: 84-7897-570-5
- GoodMan S.E.; Hedetniemi S.T. 1997. "Introduction to the Design and Analysis of Algorithms. Mc Graw Hill Book Company. United States of America. ISBN:0-07-023753-0.
- Guerrequeta G.R.; Vallecillo M.A. "Técnicas de Diseño de Algoritmos". Manuales. Universidad de Malaga.
- Troya J.M.; Capel T.M. "An Object Based Tool for Distributed Programming on Transputer Systems"
- Rossainz, Mario; Capel, Manuel. 2004. "A Parallel Programming Methodology based on High Level Parallel Compositions (CPANS)". Proceedings of XIV International Conference on Electronics, Communications, and Computers, CONIELECOMP. ISBN 0-7695-2074-X.
- Corradi A.; Zambonelli F. 1995. "Experiences toward an Object-Oriented Approach to Structured Parallel Programming". DEIS technical report no. DEIS-LIA-95-007.
- Danelutto, M.; Orlando, S; et al. "Parallel Programming Models Based on Restricted Computation Structure Approach". Technical Report-Dpt. Informatica. Università de Pisa.
- Horowitz, Sahni. 1978. "Fundamentals of Computer Algorithms". Ed. Computer Sc. Press.

AUTHOR BIOGRAPHIES



MARIO ROSSAINZ LOPEZ was born in Puebla, México and went to the University of Puebla, where he studied Sciences of Computation and obtained his degree in 1994. He works in the Faculty of Sciences of Computation of the University of Puebla from the year of 1995 where he is now in the research group of Software Engineering in the field of distributed systems. His e-mail address is: mariorl@siu.buap.mx and his Web-page can be found at <http://www.cs.buap.mx>.



MANUEL I. CAPEL TUÑÓN was born in Spain and went to the University of Granada, where he studied Physics and obtained the MSC degree in 1982. He worked in the University of Murcia before moving in 1989 to the University of Granada where he is now leading a research group in the field of Concurrent Systems. His e-mail address is: mcapel@ugr.es and his Web-page can be found at <http://lsi.ugr.es/~mcapel>.