

A Functional Package for Monitoring Branching Methods
in Combinatorial Optimization

J.P. A. Barthès

Department of Applied Mathematics and
Computer Science
University of Technology of Compiègne
60200 COMPIEGNE, France

This note announces the development of a set of computer functions for studying a wide class of combinatorial optimization problems by solving them interactively. A first implementation has been done at UTC and presently runs on a DEC PDP 11 minicomputer.

Combinatorial optimization methods such as Branch and Bound, Branch Search, etc., are used extensively because of their efficiency. Furthermore they yield good results in complex cases when sub-optimal techniques are used coupled with heuristics. It has been shown that it is possible to specify a given problem as well as the strategy to be implemented, by using a small number of parameters. Consequently, this paper presents a package of functions for implementing various branching strategies and for monitoring the search during the optimization process. The package contains a general branching mechanism which can be specialized by specifying parameters. It also provides a number of possibilities for outputting significant intermediate data or statistics.

The package may be considered a fundamental tool for the study of the interaction between the data structure and the type of strategy and in particular for the choice of heuristics for a given class of problems. This step is necessary if one wants to write efficient code for solving some classes of combinatorial optimization problems.

1. General Branching Algorithm

This paragraph is a short summary of previous work [1]. Combinatorial problems considered here consist of

- (i) a set Σ of objects called solutions
- (ii) a finite set $P = \{P_i\}$ of p properties, such that each property P_k partitions Σ into a finite number q_k ($q_k > 1$) of equivalence classes noted Σ/P_k .
- (iii) a set of feasibility conditions $C = \{C_j\}$
- (iv) a procedure which allows to extract from $\Sigma/P_1 \dots P_p$ an optimal feasible solution if there is one.

A well known representation of the search process is the search tree whose nodes represent successive examined solution classes.

Algorithm Basically the algorithm examines a solution class obtained by using some property P_i . It tries to locate an optimal solution in the class, or to determine whether or not there is any feasible solution. Possibly it computes additional information such as upper bound, lower bound, evaluation function, ... After a termination test the algorithm then goes into the process of selecting another solution class by choosing one of the previously examined classes and a new property. This property is used to obtain subclasses and one of them is selected to be examined at the next iteration.

In many places choices are made that depend on the user, who by doing so defines the strategy. They are indicated below by the qualifier *rule*, meaning a user defined procedure. For instance the *partitioning rule* corresponds to the choice of next property to be used, while the *priority rule* corresponds to the choice of the next solution class to be examined among the generated subclasses. It is worth noticing that such rules may be dynamically produced in the context of Branching Algorithms, i.e. they

may be context dependent.

The indicator and branching function mentioned in step 1.4 of the following algorithm play a crucial rule. They are used to evaluate the desirability from exploring further a given solution class and play a fundamental part in step 3.1. Actually they dictate the strategy.

The algorithm is stated in the case of a maximization problem.

Step 0

The original problem is examined first. The whole set of solutions Σ is assigned to the root of the search tree. At each iteration a solution class is examined as follows starting with Σ .

Step 1 Node Analysis

- 1.1. Check feasibility. If it is determined that the solution class does not contain any feasible solution, close the node and go to step 2.
- 1.2. Compute an upper bound for the solution class.
- 1.3. Update the state of the node. If closed (for example if terminal) then go to step 2 ; otherwise go to 1.4.
- 1.4. Compute a node indicator by evaluating the branching function. Go to step 2.

Step 2 Termination Test

Determine whether or not the search has terminated by examining the pending nodes of the search tree and by using the termination rule. If yes, then stop ; otherwise go to step 3.

Step 3 Node Generation

- 3.1. Use the pending node indicators to determine the branching node. Go to 3.2.
- 3.2. Use the partitioning and priority rules to determine the new node. Go to 3.3.
- 3.3. Update the state of the branching node and set the state of the new node to 0.

This is the end of an iteration, go to step 1 for the next iteration.

It is worth noticing that once the branching function has been defined all strategic decisions are taken in step 3 of the algorithm, while all information related to the problem data is acquired at step 1. This situation allows to write easily adequate code for implementing this type of general branching algorithm.

2. Implementation - SICOPA

General Approach SICOPA (SIMulation of Combinatorial Optimization Branching Algorithms) is a set of about 30 functions written as FORTRAN subroutines which allow the user to solve any problem that can be set up as defined in (i) through (iv) of paragraph 1. Any strategy that can be implemented by a branching function can then

specified and information about how the problem is being solved is obtained through SICOPA.

The user is left free to organize its data as he likes and must therefore provide routines for interfacing with the external world (input/output routines) as well as with SICOPA. The complexity and sophistication of those routines depend solely on the particular problem to be studied and on the user's programming skills. Generally it can be fairly simple. To illustrate this approach it suffices to give the names of the required routines which are called at various moments by SICOPA.

- . Input/Output routines
 - RDDAT reads data in
 - PRTPB prints data for checking it
 - MODDAT modifies data (optional)
 - MOVSOL moves a feasible solution into a user's defined solution area
 - WRTSOL prints part of the solution area (user controlled)
- . Search Parameters
 - INIPRM transmits search parameters to SICOPA as arguments.
- . Data Information (needed in Node Analysis Step)
 - FSULB computes upper, lower bound, optimality over solution subclass
 - BRFCN implements a branching function
- . Structural Information (needed both in Node Analysis and Node Generation Steps)
 - NXPIMI implements partitioning and priority rules
 - MAXPI returns the maximum member of generated subsets for a given property
- . Dynamic management of property area (optional)
 - PINCNT increment and decrement a property reference counter
 - PDCCNT in user's area.

Any number of additional routines may be included by the user within the limit of the machine capacity.

Once the problem has been formulated the rest is taken care of by SICOPA.

Working Modes and Available Commands. SICOPA works in two possible modes Batch or Interactive, although it was really intended to be used interactively. In batch mode SICOPA simply solves the particular problem and prints additional information such as:

- . Total number of explored nodes
- . Total elapsed time
- . Maximum number of nodes at any given time (core requirement)
- . Number of explored nodes before reaching the optimal solution
- . Maximum depth of search (interesting for complex dynamic property definition cases)
- . Display of tree width versus time (are requirement)
- . Display of tree depth versus time.

In interactive mode SICOBA works on a question/command answer basis and its possibilities can be best illustrated by giving a list of commands.

- . Exit
- . show list of commands
- . show data
- . perform single step (i.e. only node analysis for example)
- . perform n iterations
- . show current node content (examined subclass, upper-bound, lower bound, feasible solution, etc.)
- . give number of pending nodes (Instantaneous core requirement)
- . show best solution so far
- . show elapsed time
- . give number of free cells left
- . change data (user routine MODDAT)
- . start again
- . change search parameters (user routine INIPRM)
- . change tree width sampling frequency
- . change tree depth sampling frequency
- . display tree width versus time so far
- . display tree depth versus time so far
- . switch node trace flag
- . go into advanced command mode.

There is a set of about 25 advanced commands which allow the user to change pieces of information at very low level, that is to experiment on the structure. It is possible to change data but also structures (pointers) and proceed from there with standard commands. This is a dangerous but useful possibility.

In Conclusion It was found that SICOBA could be used mainly for the three following purposes

- . for testing various strategies on various combinatorial optimization problems
- . for helping to find better heuristics
- . for solving directly complex problems without using mathematical models.

Reference

1. Barthès Jean Paul A., "Branching Methods in Combinatorial Optimization", PhD Thesis, Stanford (1973).