MIT/LCS/TM-149

AN EFFICIENT ALGORITHM FOR DETERMINING THE LENGTH OF THE LONGEST DEAD PATH IN AN "LIFO" BRANCH-AND-BOUND EXPLORATION SCHEMA

Stefano Pallottino

Tommaso Toffoli

January 1980

This research was suported in part by Grant N00014-75-C-0661, Office of Naval Research, funded by DARPA, and in part by the Consiglio Nazionale delle Ticherche, Roma, Italy

AN EFFICIENT ALGORITHM FOR DETERMINING THE LENGTH OF THE LONGEST DEAD PATH IN AN "LIFO" BRANCH-AND-BOUND EXPLORATION SCHEMA*

Stefano Pallottino

Istituto per le Applicazioni del Calcolo "Mauro Picone," CNR
viale del Policlinico 137, 00161 Roma, Italy

Tommaso Toffoli

MIT Laboratory for Computer Science
545 Technology Sq., Cambridge, MA 02139

Abstract. The length of the longest dead path (LLDP) is a widely used parameter in estimating the efficiency of branch-and-bound optimization algorithms that employ the LIFO exploration schema. Thanks to two original theorems, we are able to present a particularly attractive procedure for determining of the LLDP. In fact, this procedure requires a number of storage variables which is *independent of problem size* and very small; moreover, the procedure is *self-contained* in the sense that it can be externally attached to any LIFO branch-and-bound program without interfering with its algorithms and data structures.

Keywords: Length of longest dead path, branch-and-bound, LIFO tree search.

1. Introduction

In a particular tree-exploration schema for the branch-and-bound (BAB) optimization method the search is linearized, i.e., movements are allowed only between adjacent nodes, and every arc considered in the search is eventually traversed exactly once in either direction[1]. In this way, nodes that are encountered going down the tree are encountered again—in the opposite order—on the way up; for this reason, such an exploration schema is called "last-in-first-out" (LIFO).

During an LIFO exploration, the BAB algorithm designates a certain subsequence of nodes as increasingly better candidates for the optimum. At the

*This research was supported in part by Grant N00014-75-C-0661, Office of Naval Research, funded by DARPA, and in part by the Consiglio Nazionale delle Ricerche, Roma, Italy

end of the search, the last such node represents the actual optimum. If one had sufficient foreknowledge, this node could be reached by a direct path from the tree root to the node itself; this we shall call the *true* path. The remaining explored portion can be visualized as consisting of dead subtrees (i.e., search failures) attached to the true path. The maximum departure from the true path, i.e., the length of the longest path in such dead subtrees, is a significant parameter in evaluating the efficiency of a given BAB algorithm.

While any procedure for determining the length of the longest dead path (LLDP) must somehow work in cooperation with the BAB algorithm itself, yet it would be convenient to have for this a separate, general-purpose module that can be appended —"piggy-back," as it were—to an arbitrary LIFO-oriented BAB algorithm *without* interfering with the algorithm or requiring it to manage auxiliary data structures, especially ones distributed over the tree. In the following sections we shall illustrate a simple, efficient, and self-contained procedure which (*a*) uses a finite, very small amount of storage independent of problem size, (*b*) is called in a uniform way by the BAB algorithm at every move on the tree, and (*c*) is able to tell the length of the currently longest dead path at any moment during the search and, in particular, the LLDP at the end of the search.

2. An informal illustration

Since the LLDP procedure that we are going to describe in no way affects the operation of the BAB algorithm, certain preliminary conceptual simplifications are possible. Potentially, the whole search tree is available to the BAB algorithm. At any node, according to information accumulated during the search, this algorithm is free to decide in what order to examine the outgoing arcs, and may ignore altogether the existence of any of them and, consequently, of the subtrees attached to them. On the other hand, the scope of the LLDP procedure is restricted to that portion of the problem tree which is effectively traversed. This portion is also a tree, and henceforth will be referred to simply as *the tree*. (As customary in computer science parlance, we call *tree* what in graph theory is called an *arborescence*, i.e., a rooted directed tree.) Since all routing choices are made ahead by the BAB algorithm, from the LLDP procedure's viewpoint the tree is seen as traversed in a *preassigned* order; during this traversal certain nodes are successively received as candidates for the optimum, and these candidates supersede one another in the same order as they appear.

Note that the BAB algorithm is not allowed any "look-ahead;" in other words, a node can be designated as an optimum candidate only while it is being

visited, and not at some later time. Moreover, the last candidate in the sequence is confirmed as the actual optimum only at the end of the exploration, i.e., when the search returns to the tree's root.

As we shall see, no explicit knowledge of the tree's global structure is required of the LLDP procedure, and the only information that this procedure needs to receive from the BAB algorithm is of a local nature. Namely, the LLDP procedure will be told

(*a*) when the BAB just stepped one arc down, or

(*b*) the BAB just stepped one arc up; and

(*c*) when the current node is designated as the new candidate for the optimum.

For the sake of illustration, we shall examine first a case where only one optimum candidate is eventually found. The complete exploration *journey* of Figure 1a (from START to STOP) "circumnavigates" the tree, coasting from node to node along arcs, in successive (upward or downward) *steps*, in such a way that every arc of the tree is traversed exactly once in each direction.

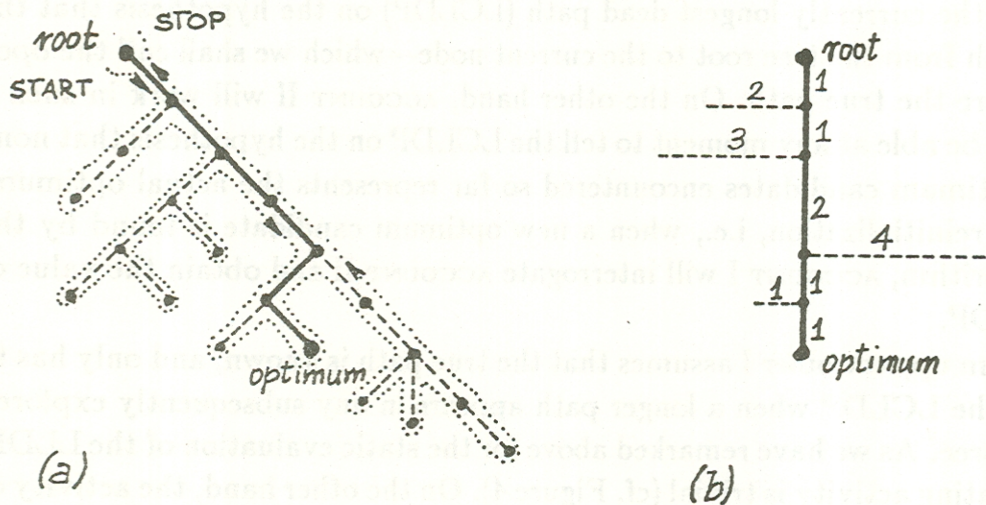

FIG. 1 *(a) Complete exploration journey (dotted line), dead subtrees (dashed), and true path (thick line). (b) Schematic representation of the dead subtrees, with their length and their position along the true path.*

Note that, since one does not know until the end of the search whether the current candidate is indeed the actual optimum, the exploration will continue—and this may create new dead subtrees—even *after* the actual optimum has been reached. Since we are interested only in the *length* of the longest dead path, the dead subtrees attached to the true path can be represented merely by the length of their longest path, as in Figure 1*b*, where for convenience the dead

subtrees explored before finding the optimum are drawn on the left, and the others on the right of the true path.

If the situation illustrated in Figure 1a were *static*, i.e., if one knew beforehand which arcs belonged to the true path, then computing the LLDP for each of the dead subtrees and determining the overall LLDP would be trivial; in this case, every time that the exploration departed from the true path and entered a dead subtree one would keep track of the distance from the root of that subtree, and update a "current maximum distance" register every time a greater value were found.

In practice, there is no way of knowing *a priori* whether the current candidate will turn out to be the actual optimum. This difficulty can be overcome by keeping two parallel accounts for dead arcs, namely, ACCOUNT I, which will proceed as if the current candidate were to "win," and ACCOUNT II as if it were to "loose." Every time the current candidate is replaced by a new one, ACCOUNT I is suitably reinitialized; both accounts then resume their independent evolution.

More explicitly, ACCOUNT I will work in such a way as to be able to tell the length of the currently longest dead path (LCLDP) on the hypothesis that the direct path from the tree root to the current node—which we shall call the *open* path—were the true path. On the other hand, ACCOUNT II will work in such a way as to be able at any moment to tell the LCLDP on the hypothesis that none of the optimum candidates encountered so far represents the actual optimum. At every reinitialization, i.e., when a new optimum candidate is found by the BAB algorithm, ACCOUNT I will interrogate ACCOUNT II and obtain the value of the LCLDP.

To sum up, ACCOUNT I assumes that the true path is known, and only has to uptdate the LCLDP when a longer path appears in any subsequently explored dead subtree. As we have remarked above for the static evaluation of the LLDP, such updating activity is trivial (cf. Figure 4). On the other hand, the activity of ACCOUNT II, which must dynamically preserve enough information about previously traversed dead subtrees to serve ACCOUNT I's reinitialization needs, is a bit more complex, and we shall discuss it in more detail below.

3. Dynamic evaluation of the length of the longest dead path

Clearly, the open path evolves dynamically during the exploration. In the example of Figure 2a, dead subtree T_3, which is "shorter" than T_2 and would be neglected in favor of the latter if the open path coincided with the true path, might become critical in the determination of the longest dead path if the

exploration were to back up along the open path, as shown in Figure 2*b*, where subtree T'_3 is "longer" than T_2.

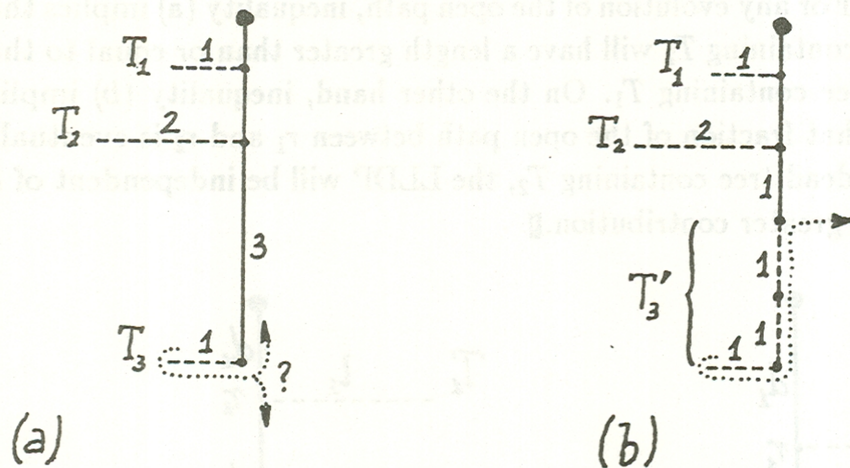

FIG. 2 *Typical situation in the dynamic evaluation of the LLDP. The open path is indicated by a solid line.*

On the other hand, independently of the future course of the exploration, subtree T_1 may be disregarded in comparison with T_2, which is already "longer" than T_1 and may only become even "longer" if the exploration were to back up above T_2's root. Thus, in this case there is no reason for ACCOUNT II to "remember" the existence of T_1; only information about root position and length of the longest path for T_2 and T_3 is still relevant at this point of the exploration and must be preserved.

In what follows, we shall formally establish general criteria for deciding what information about previously traversed dead subtrees can be discarded in the course of the search, and what information must be retained. It will turn out that at no moment does one have to carry over from previous exploration more than four independent integer quantities.

First of all, in the light of the above example it is easy to introduce the following theorem. (With reference to Figure 3, we shall call l_1, l_2, and l_3 the "lengths" of any three dead subtrees T_1, T_2, and T_3 that are attached in this order to the open path; and d_1, d_2, and d_3 the distances from each root to the previous subtree's root or, by default, to the tree's root.)

THEOREM 1 *In the situation of Figure 3a, if*

(a) $l_2 \geq l_1$,

then dead subtree T_1 *can be disregarded in the determination of the LLDP; similarly, if*

(b) $l_1 \geq l_2 + d_2$,

then T_2 can be disregarded.

Proof. For any evolution of the open path, inequality (a) implies that any dead subtree containing T_2 will have a length greater than or equal to that of any dead subtree containing T_1. On the other hand, inequality (b) implies that, no matter what fraction of the open path between r_1 and r_2 is eventually incorporated in a dead tree containing T_2, the LLDP will be independent of l_2, since l_1 will give a greater contribution.∎

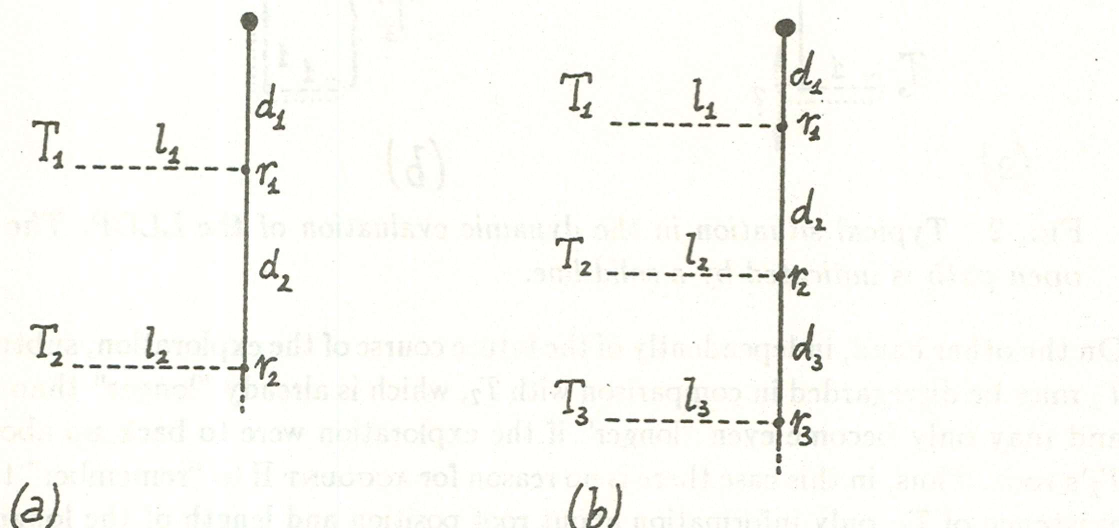

FIG. 3 *Details of tree structure and nomenclature for Theorem 1 (a) and Theorem 2 (b).*

REMARK 1 Only in the case where $l_2 < l_1 < l_2 + d_2$ will ACCOUNT II be unable to decide, without further information, whether T_1 or T_2 can be discarded.

Suppose now that ACCOUNT II, while already preserving information about two dead subtrees T_1 and T_2 (cf. Remark 1), is requested to consider a *third* dead subtree T_3. If it is impossible to discard any of the three subtrees by pairwise comparison using Theorem 1, then one of them can be discarded in any case in view of the following theorem.

THEOREM 2 *If the conditions for the applicability of Theorem 1 contemporarily fail for the pairs (T_1, T_2) and (T_2, T_3) (and, consequently, for the pair (T_1, T_3)), i.e., if $l_2 < l_1 < l_2 + d_2$ and $l_3 < l_2 < l_3 + d_3$, then dead subtree T_2 can be disregarded in the determination of the LLDP.*

Proof. As the search progresses, eventually part of the open path will be incorporated into dead subtrees, while the rest will remain in the true path (TP). Considering the root r_2 of subtree T_2, two cases are possible:

(a) $r_2 \in \mathrm{TP}$;

in this case, since $l_1 > l_2$, then dead subtree T_2 can be discarded.

(b) $r_2 \notin \mathrm{TP}$;

in this case, since $l_2 < l_3 + d_3$, then dead subtree T_2 can be discarded.∎

REMARK 2 At no moment during the exploration does ACCOUNT II have to retain more information about previously traversed dead subtrees than that represented by four integers, namely, the root position and the length of two particular dead subtrees.

Observe that, owing to its "pessimistic" attitude, ACCOUNT II never needs to be corrected or reinitialized when a a new optimum candidate is encountered. On the other hand, owing to its "optimistic" attitude, ACCOUNT I may often prove wrong and must be reinitialized when a new optimum candidate is encountered. The only information that is required for this reinitialization is, of course, the length of the currently longest dead path (LCLDP), which on such occasion coincides with the value of l_1 in ACCOUNT II.

At the end of the BAB exploration, when the current optimum candidate is indeed the actual optimum, the LCLDP in ACCOUNT I (denoted by l in Figure 4) will coincide with the LLDP, i.e., with the quantity the we set out to determine.

4. The LLDP procedure *DIP*

As noted in Section 2, the LLDP procedure that we have been discussing is called by the BAB algorithm every time the latter steps up or down or finds a new optimum candidate. For convenience of implementation, the explicit version of this procedure that we present below under the name of *DIP* will assume that such calls have been lumped into groups each corresponding to a *dip*, i.e., an uninterrupted sequence of "down" calls followed by an uninterrupted sequence of "up" calls; an "optimum" call may appear between the two sequences, and either sequence may be empty.

Procedure *DIP* will be called according to the format

call *DIP(down, find, up)*,

where the integer variables *down* and *up* represent respectively the number of downward and upward steps in a particular dip, and the logical variable *find* assumes the value **true** if a new optimum candidate was found at the bottom of that dip, and **false** otherwise.

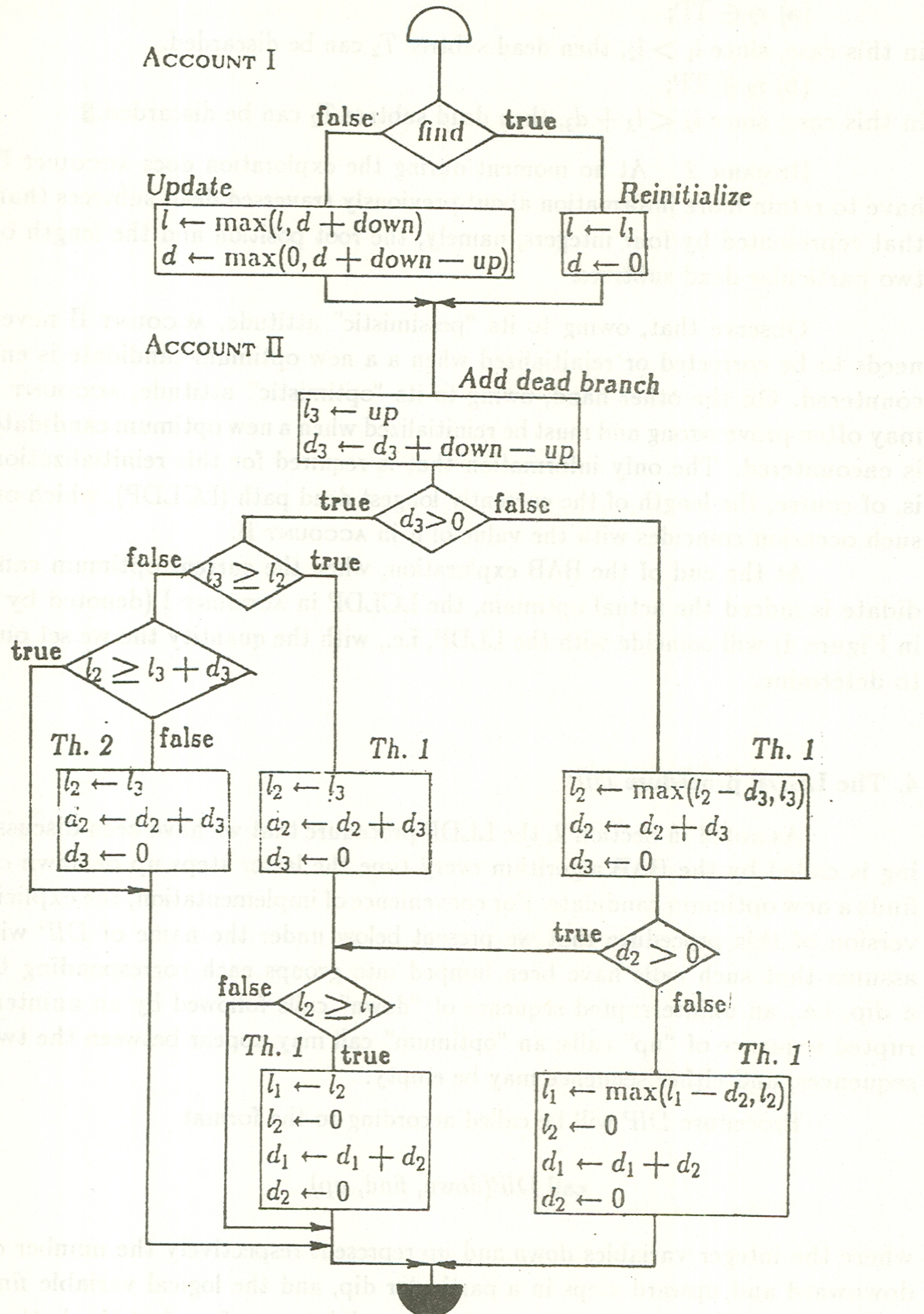

FIG. 4 ***Overall structure of the LLDP procedure DIP.***

The procedure itself is illustrated in Figure 4. All internal variables are of type **own integer** and are set to 0 at the beginning of the search. Variable *up* is renamed "l_3" in ACCOUNT II only to put in better evidence the regular structure of the algorithm. In order to simplify the program's structure, subscript 3 is always associated with the subtree corresponding to the current dip. The variables associated with the other two subscripts (1 and 2) are used as needed to carry over relevant information about at most two previously encountered dead subtrees, and are set to 0 when not in use.

A FORTRAN listing of procedure *DIP* together with comments and examples can be obtained by writing to either of the two authors.

Reference

[1] Giorgio Gallo, Peter L. Hammer, *and* Bruno Simeone, "Quadratic Knapsack Problems," *IX Int. Symp. on Math. Progr.*, Budapest, August 1976.