Composite version of B&B algorithm: experimental verification of the efficiency

V O Groppen\textsuperscript{1} and A A Berko

Informatics and Computer Sciences Department,\textit{ North Caucasian Mining and Metallurgical Institute}, 44 Nikolaev Street, 362021, Vladikavkaz, Russia

\textsuperscript{1}E-mail: groppen@mail.ru

Abstract The aim of this paper is to verify the efficiency for a composite version of branch and bound algorithm combining B&B branching strategy and bounds computing with the pruning rules used in dynamic programming for selecting of direction of movement by a search tree. As criteria of efficiency we used random access memory (RAM) and time gains of finding integer programming problem solution by a composite version of branch and bound algorithm in comparison with its traditional implementation. The number of variables in the problems and restrictions did not exceed 25 and 15, respectively. Experimental verification have shown that for solving time used as a criterion of the efficiency, the advantages of composite version are beyond doubt compared to the used RAM value as criterion often taking the side of the classical B&B algorithm.

1. Introduction

The branch-and-bound (B&B) as well as branch and cut (B&C) algorithms are fundamental and widely-used methodologies for producing exact solutions to NP-hard optimization problems. The B&B technique was proposed by A. H. Land and A. G. Doig in 1960 [1], whereas the most popular B&C technology – dynamic programming, was developed in 1954 by R. Bellman [2,3]. Each of these two methodologies includes different types of four algorithmic components: bounds computing technique, the search strategy, the branching strategy, and the pruning rules whereas in [4] only three of these components were proposed. The main advantage of these methods is the guaranty to obtain a globally optimal solution, but their disadvantage in the conditions of a lack of decision-making time is the exponential dependence of the running time on dimension of the problem solved. This explains the large number of diverse attempts to reduce the running time of these algorithms [5-7].

In contrast to the technique proposed in [8] for B&B procedures aimed at running time reduction, it is not possible to evaluate analytically the efficiency of composite versions of B&B algorithms - approach proposed in [9,10]. The thing is that there are two opposite processes inherent in it: on the one hand, it increases the time spent at each iteration for analyzing of each possible direction of descending by a search tree, and on the other hand, due to pruning of obviously “unpromising” directions, this approach reduces the time spent to choose such a direction.

The aim of this work is in the experimental verification of the effectiveness of combining technologies inherent in B&B methods and dynamic programming for finding of globally optimal solutions to discrete optimization problems. In the experimental verification below of the proposed in [9,10] such a composite algorithm, procedure selecting the directions of descent on a search tree created by B&B technique is supplemented by "bad" vectors of variables pruning technology,
developed for dynamic programming. As we study the comparative efficiency of classical and composite B&B procedures searching for a globally optimal solution of extreme problems with Boolean variables, used are the following symbols, assumptions and definitions.

2. Notation, assumptions and definitions
Below is analyzed comparative efficiency of two kinds of branch and bound algorithms searching solutions of single-criterion problems with Boolean variables of the following form:

\[
\begin{align*}
F &= \sum_i C_i z_i \rightarrow \max(\min); \\
\forall j : \sum_i b_{i,j} z_i \theta_j a_j; \\
\forall i : z_i = 1,0; \\
\forall j : \theta_j \in \{\leq,=,\geq\},
\end{align*}
\]

where \( F \) is either minimized or maximized goal function; \( C_i \) and \( b_{i,j} \) - coefficients, \( a_j \) - constants, and here we assume that all coefficients and constants are non-negative. Nevertheless, all numerical examples below illustrating the work of the analyzed algorithms correspond to the knapsack problem \cite{11} - system (1) case, where \( F \) is maximized, \( j = 1, \theta_1 = "\leq" \):

\[
\begin{align*}
F &= 7z_1 + 3z_2 + 5z_3 + 9z_4 + 2z_5 \rightarrow \max; \\
3z_1 + 4z_2 + 2z_3 + 7z_4 + 6z_5 &\leq 10; \\
\forall i : z_i &\in \{1,0\}.
\end{align*}
\]

Further we seek solutions on a set of search trees, applying the following definitions:
1. The search tree \( G(X, U) \), where \( X \) is the set of vertices, \( U \) – the set of arcs, is constructed during the search for system (2) solution.
2. "Hanging" vertex at any iteration is considered to be a vertex of the constructed search tree \( G(X, U) \), which is free of outcoming arcs. A set of hanging vertices is below denoted as \( X_T \). The root vertex of the search tree \( x_0 \in X \) is considered as "hanging" vertex at the first iteration.
3. The upper bound \( \Delta(x_k) \) of value \( F \), corresponding to the vertex \( x_k \in X \) which is associated with vector of variables of the problem (2), is further defined as follows: \( \Delta(x_k) \) value is equal to \( \sum_{i \in I(x_k)} C_i z_i + \sum_{j \in I_1(x_k)} C_j \) if system (2) constraint is satisfied and equal to \( -\infty \) otherwise, where \( I \) is the set of indices of all variables, whereas \( I_1(x_k) \) is the subset of indices, corresponding to the included in the basis variables, associated on the search tree with the vertex \( x_k \).
4. Similarly is defined the value of the lower bound \( \delta(x_k) \) of \( F \) value:

\[
\delta(x_k) = \sum_{i \in I_1(x_k)} C_i z_i, \quad \text{if} \quad \sum_{i \in I_1(x_k)} b_{i,j} z_i \leq 10, \quad \text{and} \quad \delta(x_k) \text{is equal to} \quad -\infty \quad \text{otherwise}.
\]

One of the distinctive features of the composite version of B&B algorithm is assignment of vector \( R(x_k) = \{r_1(x_k), r_2(x_k), \ldots, r_{m+2}(x_k)\} \) to each hanging vertex \( x_k \in X \) of the search tree \( G(X, U) \) instead of \( \Delta(x_k) \). First two components of this vector are equal to values of described above \( \Delta(x_k) \) and \( \delta(x_k) \), and the rest are determined according to the expression:

\[
\forall m + 2 \geq j > 2 : r_j(x_k) = a_{j-2} - \sum_{q \in I_1(x_k)} b_{q,j-2} z_q.
\]
This allows us to use in such algorithms the procedure for cutting off “bad” plans used in dynamic programming: if between hanging vertices of a search tree \( G(X, U) \) there are two vertices \( x_k \) and \( x_q \), for which true is at least one the following conditions:

\[
I_1(x_k) = I_1(x_q); \\
\text{either } \delta(x_k) \geq \delta(x_q), \text{if } F \rightarrow \text{max}, \text{or } \delta(x_k) \leq \delta(x_q), \text{if } F \rightarrow \text{min}; \\
\forall j > 2: \text{either } r_j(x_k) \leq r_j(x_q) \text{ if } \theta_{j-2} = \geq, \text{ or } r_j(x_k) \geq r_j(x_q) \text{ if } \theta_{j-2} = \leq; \\
\text{either } \delta(x_k) \geq \Delta(x_q), \text{if } F \rightarrow \text{max}, \text{or } \Delta(x_k) \leq \delta(x_q), \text{if } F \rightarrow \text{min}; \\
\forall j > 2: \text{either } r_j(x_k) \leq r_j(x_q) \text{ if } \theta_{j-2} = \geq, \text{ or } r_j(x_k) \geq r_j(x_q) \text{ if } \theta_{j-2} = \leq; \\
\frac{1}{2} \geq \forall j r_j(x_q) \leq r_j(x_k), \text{if } \theta_{j-2} = \geq, \text{ or } r_j(x_q) \geq r_j(x_k) \text{ if } \theta_{j-2} = \leq;
\]  

(4) 

where vertex \( x_q \) and all corresponding to it problem (1) planes can be ignored.

It should be noted that the cut-off vertex \( x_q \) procedure, implemented by system (5), is absent in the B&B methods, as well as in dynamic programming. It was made possible only with the combination of these two approaches. As shown below we also believe that problem (1) solving time by any kind of B&B algorithm in the first approximation linearly depends on the number of vertices of constructed search tree \( G(X, U) \). A common feature of the presented below images of such trees, illustrating search of system (2) solutions, is the encoding of the vertices corresponding to variables equal to “1” by the gray color, whereas white vertices correspond to the values of zero equal variables. Vertex of the last tier with "fat" contour corresponds to the optimal vector of variables \( Z \).

Whereas in the following sections 3 and 4 are described the two B&B type methods adapted for knapsack problem solving and conditions of experiments connected with problem (1) solving, content of the section 5 reflects results of these experiments received by the two B&B algorithms.

### 3. Branch and Bound algorithm

Description of such algorithm is below represented by a consistent construction of the search tree \( G(X, U) \):

#### Algorithm 1

Step 1. On the set of hanging vertices \( X_1 \subseteq X \) of the built search tree \( G(X, U) \) is selected vertex \( x_j \) with the best upper bound. If this is done on the first iteration, then this vertex a priori corresponds to be the root vertex of this tree.

Step 2. If the selected vertex meets equality \( I_j = I \), then go to the step 5, otherwise - to the next step.

Step 3. The branching is made from the vertex \( x_j \), which was selected at the first step of the latest iteration. A new set of hanging vertices of the tree again is denoted as \( X_1 \).

Step 4. For each new vertex \( x_k \in X_1 \) belonging to the "bush" which was built at the previous step is computed bound \( \Delta(x_k) \). Go to the first step.

Step 5. The algorithm is complete. The vector of variables corresponding to the selected during the first step of the latest iteration vertex of the search tree is optimal.

Problem (2) solution by algorithm 1 leads to the step by step construction of the search tree \( G_j(X_j, U_j) \), shown below in figure 1. According to this tree optimal vector of variables of the problem (2), \( Z = \{1, 0, 0, 1, 0\} \), corresponding value of the objective function \( F = 16 \), the number of vertices of the search tree \( G_j(X_j, U_j) \), \( \vert X_j \vert = 27 \), the upper bound of the number of hanging vertices is equal to 14. Since each hanging vertex of a search tree corresponds to one bound, it can be argued that the amount of RAM used is directly proportional to the maximum value of \( \vert X_j \vert \).
4. Composite implementation of branch and bound method

During this approach realization, each vertex $x_j$ of the search tree $G_2(X_2, U_2)$ created by this algorithm, is associated with described above vector $R(x_j)$. Below is presented a complete but not formal description of composite implementation of algorithm 1, where cutting off vertices, satisfying (4) or (5) conditions, is substituted by their crossing out.

**Algorithm 2**

Step 1. On the set of not crossed out hanging vertices $X_2^T \subseteq X_2$ of the constructed search tree $G_2(X_2, U_2)$ is selected vertex $x_0 \in X_2$ with the "best" first component of vector $R(x_0)$. If this is done on the first iteration, then this vertex a priori is considered to be the root vertex $x_0$ of this tree.

Step 2. If for the selected vertex true is the equality $I_1 = I$, then go to the step 7, otherwise - to the next step.

Step 3. Created is a “bush” with the root vertex coinciding with selected at the first step of the latest iteration vertex $x_j$. A new set of hanging vertices of the tree we again denote as $X_2^T \subseteq X_2$.

Step 4. We calculate vector $R(x_j)$ for each hanging vertex $x_j \in X_2^T$ of the built in the previous step “bush”. The first two components of this vector are equal to $A(x_j)$ and $\delta(x_j)$, the other components are calculated according to (3).

Step 5. If in the set of vertices $X_2^T$ there are two vertices $x_k$ and $x_q$, satisfying either (4) or (5), then vertex $x_q$ is crossed out.

Step 6. Go to the step 1.

Step 7. The algorithm is compete. The vector of variables corresponding to the selected at the second step of the latest iteration vertex is optimal.

Below in the figure 2 there is presented the search tree $G_1(X_1, U_1)$ built by the algorithm 1 solving problem (2).

**Figure 1.** Search tree $G_1(X_1, U_1)$ built by the algorithm 1 solving problem (2).

**Figure 2.** Search tree $G_2(X_2, U_2)$ built by the algorithm 2 solving problem (2).
corresponds to three numbers, and the maximum number of hanging vertices of this tree is equal to 16, the ratio \( \eta_2 = \frac{|X^T_2|}{3 |X^T_2|} = 0.67 \) demonstrates the growth of the volume used by the algorithm 2 RAM compared to algorithm 1.

Data of the hardware and software used are presented in the Table 1 below.

**Table 1. Parameters of used hardware and software.**

| Computer settings                      | Software options     |
|----------------------------------------|----------------------|
| Processor manufacturer                | Intel                |
| CPU Type                              | Core i5-7300HQ       |
| CPU frequency upper bound             | 2.50 GHz             |
| Cache memory volume                   | 6 MB                 |
| RAM                                    | 16 GB                |
| Memory Type                           | DDR4-2400            |
| Memory frequency                      | 1200 MHz             |
| Graphics Card Manufacturer            | NVIDIA               |
| Graphics controller                   | GeForce GTX 1050     |
| HDD                                   | 1024 GB              |
| Operating system                      | Windows 10 PRO (64 bit) |
| Programming language                  | C#                   |
| The size of the software that implements both algorithms | 25 Kb                |

In the course of the experiments, each of presented above algorithms solved more than 3000 problems, the number of problem (1) variables “\( n \)” varied in the range of 2–25, the number of restrictions “\( m \)” varied in the range of 1–15, ten problems with randomly assigned integer constants in the range 1 – 100 (uniform distribution) corresponded to each combination “\( n \)” and “\( m \)”. For each such a combination, the upper bound, lower bound, and average value of solving problem (1) time and RAM by each of the algorithms above were recorded. Corresponding experimental dependences are presented in the next section.

5. Results of experiments

The figure 3 there are experimental dependences of algorithm 1, showing upper (red), average (green) and lower (blue) values of running time \( T_1 \) on problem (1) “\( n \)” and “\( m \)” values. Similar dependences of algorithm 2 running time \( T_2 \) are presented at figure 4.
The time gain $\eta_1$ of finding problem (1) solution by algorithm 2 in comparison with algorithm 1 is determined as follows:

$$\eta_1 = \frac{T_1(m_1, n_1)}{T_2(m_2, n_2)}, \quad (6)$$

where $m_1 = m_2$; $n_1 = n_2$. Dependencies of the upper bound, the average value and the minimum value $\eta_1$ on the values "m" and "n", constructed on the basis of (6) and according to the experimental dependencies presented above in figures 3 and 4, are shown in figure 5 below.

If not time of solving problem (1), but the amount of RAM used is chosen as a criterion for the efficiency of the algorithms 1 and 2, then the advantage of the composite version of B&B becomes not so obvious. At figure 6 below, denoted as RAM$_1$ are presented experimental dependences of algorithm 1, reflecting upper (red), average(green) and lower (blue) values of used RAM on problem (1) "n" and "m" values. Similar dependences for used by algorithm 2 RAM, denoted as RAM$_2$, are presented at figure 7.
The used RAM gain $\eta_2$ of finding problem (1) solution by algorithm 2 in comparison with algorithm 1 is determined as follows:

$$\eta_2 = \frac{\text{RAM}_1(m_1, n_1)}{\text{RAM}_2(m_2, n_2)},$$

(7)

where $m_1 = m_2; n_1 = n_2$.

Dependencies of the upper bound (red), the average (green) value and the minimum (blue) value $\eta_2$ on the values "m" and "n", constructed on the basis of (7) and according to the experimental dependencies presented above in figures 6 and 7, are shown in figure 8 below.

In figure 9 the "green" area shows cases of average value of RAM$_2$ exceeding average RAM$_1$ (red) value where, according to the RAM criterion, the traditional B&B version is more efficient than the composite one.

Conclusions
In the case of running time criterion, the advantages of using the composite version of the B&B method are beyond doubt due to the following results of experimental verification:

- In the course of the experiments the average value of the ratio $\eta_1 = T_1/T_2$ was equal to 39, whereas the lower bound of $\eta_1$ was close to one. The efficiency of the composite version of the B&B algorithm in comparison with the traditional one is the higher, the "worse" task is.

In the case of RAM used as criterion, the advantage of the composite version of B&B is not so obvious: the upper bound of the ratio $\eta_2$ in the course of the experiments was equal to 229.3 (n = 24, m...
whereas the lower bound of $\eta_2$ was equal to 0.0444 ($n = 3$, $m = 14$) and moreover, the area in which $\eta_2$ did not exceed one was quite significant. The latter means that before choosing the version of solving problem (1) B&B algorithm, it is useful to select its effectiveness criterion.

References
[1] Land A H and Doig A G 1960 An automatic method of solving discrete programming problems, *Econometrica* **28**, 3497
[2] Hugues M, Marchand H, Martin A, Weismantel R and Wolsey L 2002 Cutting planes in integer and mixed integer programming, *Discrete Applied Mathematics* **123** 387
[3] Bellman R 1954 The Theory of Dynamic Programming (*Santa Monica: The RAND Corporation*)
[4] Morrison D R, Jacobson S H, Sauppe J J and Sewell E C 2016 Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization* **19** 79
[5] Derpich I and Sepúlveda J M 2017 Accelerating the B&B algorithm for integer programming based on flatness information: An approach applied to the multidimensional knapsack problem. *Croatian Operational Research Review* **7**, 1119
[6] Carneiro T, Muritiba A E, Negreiros M and de Campos G A L 2011 A new parallel schema for branch-and-bound algorithms using GPGPU. *Proc. of the 23rd International Symposium on Computer Architecture and High Performance Computing* (Vitória) (Vitória-Brazil: IEEE) p. 41–47.
[7] Bendjoudi A, Chekini M, Gharbi M and Mehdi M 2013 "Parallel B&B Algorithm for Hybrid Multi-core/GPU Architectures," *Proc. IEEE 10th Int. Conf. High Performance Computing and Communications & IEEE Int. Conf. on Embedded and Ubiquitous Computing, (Zhangjiajie)* (Zhangjiajie: IEEE) pp. 914-921.
[8] Borisenko A B and Gorlatch S 2018 Parallel hybrid metaheuristics approach for optimal selection of production equipment. *Bulletin of Tambov State University, 24*, 2, 228.
[9] Groppen V O, Datiev A A and Panovskaja K V 2018 *Proc. Seminar “IT-technologies: Theory and Practice”,* Issue of the North-Caucasian Institute of Mining and Metallurgy (Vladikavkaz: Vesta) pp 3 – 20 [in Russian]
[10] Groppen V O 2017 *Proc. Seminar “IT-technologies: Theory and Practice”,* Issue of the North-Caucasian Institute of Mining and Metallurgy (Vladikavkaz: Terek) pp 21–28 [in Russian]
[11] Caccetta L and Kulanoot A 2001 Computational Aspects of Hard Knapsack Problems. *Nonlinear Analysis* **47** 5547 doi:10.1016/s0362-546x(01)00638-7.