Investigation of selection strategies in branch and bound algorithm with simplicial partitions and combination of Lipschitz bounds

Remigijus Paulavičius · Julius Žilinskas · Andreas Grothey

Received: 5 October 2009 / Accepted: 19 October 2009 / Published online: 8 November 2009 © Springer-Verlag 2009

Abstract  Speed and memory requirements of branch and bound algorithms depend on the selection strategy of which candidate node to process next. The goal of this paper is to experimentally investigate this influence to the performance of sequential and parallel branch and bound algorithms. The experiments have been performed solving a number of multidimensional test problems for global optimization. Branch and bound algorithm using simplicial partitions and combination of Lipschitz bounds has been investigated. Similar results may be expected for other branch and bound algorithms.

Keywords  Global optimization · Branch and bound · Selection strategies · Lipschitz optimization · Parallel branch and bound

1 Introduction

Many problems in engineering, physics, economics and other fields may be formulated as optimization problems, where the maximum value of an objective function must be found. Mathematically the problem is formulated as
where the objective function \( f(x) \), \( f : \mathbb{R}^n \rightarrow \mathbb{R} \), is a nonlinear function of continuous variables, \( D \subset \mathbb{R}^n \) is a feasible region, \( n \) is the number of variables. Besides the global optimum \( f^* \) one or all global optimizers \( x^* : f(x^*) = f^* \) must be found.

Branch and bound is a technique for the implementation of covering global optimization methods as well as combinatorial optimization algorithms. An iteration of a classical branch and bound algorithm processes a node in the search tree representing a not yet explored subspace of the solution space. The iteration has three main components: selection of the node to process, branching of the search tree and bound calculation. Branching is implemented by division of the subspaces. The algorithm detects subspaces, which cannot contain a global optimizer, by evaluating bounds for the optimum over considered subspaces. Subspaces which cannot contain a global maximum are discarded from further search pruning the branches of the search tree. The rules of selection, branching and bounding differ from algorithm to algorithm.

The main strategies of selection are:

- **Best first** – select a candidate with the maximal upper bound. The candidate list can be implemented using heap or priority queue.
- **Depth first** – select the youngest candidate. A node with the largest level in the search tree is chosen for exploration. A FILO structure is used for the candidate list which can be implemented using a stack. In some cases it is possible to implement this strategy without storing of candidates, as it is shown in [19].
- **Breadth first** – select the oldest candidate. All nodes at one level of the search tree are processed before any node at the next level is selected. A FIFO structure is used for the candidate list which can be implemented using a queue.
- **Improved selection** – based on heuristic [3,10], probabilistic [5] or statistical [18, 20] criteria. The candidate list can be implemented using heap or priority queue.

Node selection rules influence the efficiency of the branch and bound algorithm and the number of nodes kept in the candidate list. The goal of this paper is to experimentally investigate the influence of selection strategies to the performance of sequential and parallel algorithms. Although the experiments have been performed on a particular algorithm described in the next section, similar features may be expected in other branch and bound algorithms as well.

2 Branch and bound with simplicial partitions and improved combination of different bounds for Lipschitz optimization

Lipschitz optimization is one of the most deeply investigated subjects of global optimization. It is based on the assumption that the slope of an objective function is bounded. The advantages and disadvantages of Lipschitz global optimization methods are discussed in [7,8]. A function \( f : D \rightarrow \mathbb{R}, \ D \subset \mathbb{R}^n \), is said to be Lipschitz if it satisfies the condition

\[
|f(x) - f(y)| \leq L\|x - y\|, \quad \forall x, y \in D, \tag{1}
\]
where $L > 0$ is a constant called Lipschitz constant, $D$ is compact and $\| \cdot \|$ denotes a norm. The Euclidean norm is used most often in Lipschitz optimization, but other norms can also be considered.

Although hyper-rectangular partitions are usually used in global optimization, other types of partitions may be more suitable for some specific problems. In this paper we use simplicial branch and bound with combination of Lipschitz bounds. Advantages and disadvantages of simplicial partitions are discussed in [21]. Since a simplex is a polyhedron in $n$-dimensional space with the minimal number of vertices, simplicial partitions are preferable when the values of an objective function at the vertices of partitions are used to compute bounds. Otherwise values at some of the vertices of hyper-rectangular partitions may be used [17]. However, for simplicial branch and bound, the feasible region should be initially covered by simplices. The most preferable initial covering is face to face vertex triangulation—partitioning of the feasible region into finitely many $n$-dimensional simplices, whose vertices are also the vertices of the feasible region. We use a general (any dimensional) algorithm for combinatorial vertex triangulation of hyper-rectangle [21] into $n!$ simplices. Simplices are subdivided into two by a hyper-plane passing through the middle point of the longest edge and the vertices which do not belong to the longest edge.

In Lipschitz optimization the upper bound for the optimum is evaluated exploiting the Lipschitz condition

$$f(x) \leq f(y) + L\|x - y\|.$$ 

It is known that

$$|f(x) - f(y)| \leq L_p\|x - y\|_q,$$ 

where $L_p = \sup \{\|\nabla f(x)\|_p : x \in D\}$ is the Lipschitz constant, $\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$ is the gradient of the function $f(x)$ and $1/p + 1/q = 1$, $1 \leq p, q \leq \infty$.

In the present paper the Lipschitz constants vary substantially, from 1 to over 2,639,040. These constants have been evaluated with a very fine grid search algorithm on 1,000$^n$ points for $n$-dimensional test problems and are thus very close to the smallest possible ones. Optimal values are given in [14].

If the bounds over a simplex are evaluated using the function values at the vertices, the lower bound for the optimum is the largest value of the function at a vertex:

$$LB(I) = \max_{x_v \in V(I)} f(x_v),$$

where $V(I)$ is the set of vertices of the simplex $I$. A combination of the upper bounds based on the extreme (infinite and first) and Euclidean norms over a multidimensional simplex was proposed and investigated in [14]:

$$UB_C(I) = \min_{x_v \in V(I)} \{f(x_v) + K(x_v)\},$$

(3)
where

\[ K(x_v) = \min \left\{ L_1 \max_{x \in I} \|x - x_v\|_\infty, L_2 \max_{x \in I} \|x - x_v\|_2, L_\infty \max_{x \in I} \|x - x_v\|_1 \right\}. \]

An improved upper bound with the first norm was proposed in [15]:

\[ UBF(I) = \max_{x \in I} \left( \min_{x_v \in V(I)} \{ f(x_v) + L_\infty \|x - x_v\|_1 \} \right). \]  (4)

However, the first norm does not always give the best bounds [15]. In some cases combinations of bounds (3) may give better results. Therefore, the improved combination [16] is used, where the improved bound (4) is combined with the combination of bounds based on the infinite and Euclidean norms (simpler bound based on the first norm is not used in the combination since improved bound is based on it):

\[ UB(I) = \min \{ UB_C(I), UBF(I) \} \]
\[ = \min \left\{ \min_{x_v \in V(I)} \{ f(x_v) + K'(x_v) \}, \max_{x \in I} \left( \min_{x_v \in V(I)} \{ f(x_v) + L_\infty \|x - x_v\|_1 \} \right) \right\}, \]  (5)

where

\[ K'(x_v) = \min \left\{ L_1 \max_{x \in I} \|x - x_v\|_\infty, L_2 \max_{x \in I} \|x - x_v\|_2 \right\}. \]

It is also promising to develop improved bounds for other norms.

Apart from the standard best first, depth first and breadth first strategies, statistical selection has been implemented. In this strategy the candidate with the maximal criterion value [20]

\[ \tilde{u}(I) = -\left( f^* - \frac{1}{n+1} \sum_{x_v \in V(I)} f(x_v) \right)^2 - \left( \max_{x_v \in V(I)} f(x_v) - \frac{1}{n+1} \sum_{x_v \in V(I)} f(x_v) \right)^2 \]
\[ = \min_{x_v \in V(I)} \left\| x_v - \frac{1}{n+1} \sum_{x_v \in V(I)} x_v \right\|_2 \]

is chosen where \( f^* \) is the global maximum or the upper bound for it. In the developed algorithms heap structure is used for the candidate lists when best first and statistical selection strategies are used.

3 Experimental investigation of selection strategies

In this section the results of computational experiments are presented and discussed. Various test problems \((n \geq 2)\) for global optimization from [7,9,11] have been used.
in our experiments. Test problems with \((n = 2\) and \(n = 3\)) are numbered according to [7] and [11]. For \((n \geq 4\)) problem names from [9,11] are used.

Computational experiments have been performed on the computer cluster Ness at Edinburgh Parallel Computing Center (EPCC). It consists of a cluster of two SMP boxes that form the back-end: 2.6 GHz AMD Opteron (AMD64e) processors with 2 GB of memory (32 processors divided into two 16-processors boxes), and a two-processor front-end. The computer cluster runs Linux operating system (Scientific Linux) and Sun Grid Engine.

3.1 Results of sequential branch and bound algorithm

In this section the sequential branch and bound algorithm for global optimization with simplicial partitions and combination of Lipschitz bounds has been investigated. The results of different selection strategies have been compared. The numbers of function evaluations \((f_{\text{eval}})\) and execution time \((t(s))\) using different selection strategies are shown in Table 1. The average numbers of function evaluations \((\bar{f}_{\text{eval}})\) and average execution time \((\bar{t}(s))\) for different dimensionality test problems are shown in Table 2.

For \(n = 2\)-dimensional test problems the \textit{depth first} selection strategy is the least efficient. For test problems with higher dimensionality \((n \geq 3)\) the average number of function evaluations are very similar for all selection strategies and the differences are insignificant. For test problems of all dimensionalities the smallest execution time is achieved when \textit{depth first} and \textit{breath first} selection strategies are used, despite the fact that sometimes the number of function evaluations is higher. A possible reason is that the time required for insertion and deletion of elements to/from non-prioritized structure does not depend on the number of elements in the list. \textit{Best first} and \textit{statistical} selection strategies require prioritized list of candidates, and even with heap structure insertion is time consuming when the number of elements in the list is large.

The progress of search to locate the global solution is estimated using the ratio

\[
\frac{\text{runtime}}{\text{evaluations}} = \frac{f_{\text{eval}}(f^*)}{\bar{f}_{\text{eval}}} ,
\]

where \((f_{\text{eval}}(f^*))\) is the number of function evaluations after which the best global solution \(f^*\) is found and \((f_{\text{eval}})\) is the number of function evaluations during the whole optimization period. This value is between zero and one and shows how fast the global solution \(f^*\) is found during the optimization process. The ratios \((r(f^*))\) for all test problems are shown in Fig. 1. The average ratios \((\bar{r}(f^*))\) for test problems of different dimensionalities are shown in Table 3. For almost all test problems the smallest ratio is achieved when \textit{statistical} selection strategy is used and average ratios are more than two times smaller for this strategy comparing with other selection strategies. For test problems of dimensionalities \(n = 2\) and \(n = 3\) the ratios are very similar for \textit{best first} and \textit{depth first} search strategies, but for \(n \geq 4\) better results (smaller ratio) are achieved when \textit{depth first} selection strategy is used. For almost all test problems the worst (biggest) ratios are achieved when \textit{breadth first} selection strategy is used.
The total numbers of simplices ($TNS$) and the maximal sizes of candidate list ($MCL$) at the search tree for different selection strategies are shown in Table 4. The average total numbers of simplices ($\overline{TNS}$) and the average maximal sizes of candidate list ($\overline{MCL}$) are shown in Table 5. For $n = 2$-dimensional test problems ($TNS$) is largest when depth first selection strategy is used. For higher dimensionality ($n \geq 3$) the
Investigation of selection strategies in branch and bound algorithm

Table 2  Average numbers of function evaluations and execution time for different selection strategies

<table>
<thead>
<tr>
<th>n</th>
<th>Best first</th>
<th>Statistical</th>
<th>Depth first</th>
<th>Breadth first</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f_{eval}$</td>
<td>$t(s)$</td>
<td>$f_{eval}$</td>
<td>$t(s)$</td>
</tr>
<tr>
<td>2</td>
<td>179</td>
<td>1</td>
<td>194</td>
<td>1.1</td>
</tr>
<tr>
<td>3</td>
<td>664</td>
<td>0.5</td>
<td>678</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>2,924</td>
<td>1.0</td>
<td>3,019</td>
<td>1.0</td>
</tr>
<tr>
<td>5–6</td>
<td>1,217,064</td>
<td>16.0</td>
<td>1,220,066</td>
<td>15.0</td>
</tr>
</tbody>
</table>

Fig. 1  The ratios $r(f^*)$ for the algorithms with different selection strategies

Table 3  Average ratios $r(f^* / TNS)$ for the algorithms with different selection strategies

<table>
<thead>
<tr>
<th>n</th>
<th>Best first</th>
<th>Statistical</th>
<th>Depth first</th>
<th>Breadth first</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.47</td>
<td>0.20</td>
<td>0.47</td>
<td>0.63</td>
</tr>
<tr>
<td>3</td>
<td>0.27</td>
<td>0.09</td>
<td>0.26</td>
<td>0.52</td>
</tr>
<tr>
<td>4</td>
<td>0.18</td>
<td>0.05</td>
<td>0.14</td>
<td>0.32</td>
</tr>
<tr>
<td>5–6</td>
<td>0.19</td>
<td>0.00</td>
<td>0.03</td>
<td>0.21</td>
</tr>
</tbody>
</table>

values of $(TNS)$ are very similar for all selection strategies. But the maximal candidate list $(MCL)$ at the search tree varies significantly depending on selection strategies. The best results (the smallest $(MCL)$) achieved when depth first selection strategy is used and it is up to $\sim 7,000$ times smaller than $(MCL)$ with other selection strategies. The maximal candidate list $(MCL)$ is largest when breadth first selection strategy is used. When statistical selection strategy is used $(MCL)$ is up to $\sim 5$ times smaller than when best first strategy is used. This explains why execution time is smaller when statistical selection strategy is used. This is because the time required for insertion and deletion of candidates to/from heap structure depends on the number of elements in the heap.
3.2 Results of parallel branch and bound algorithm

Global optimization algorithms are computationally intensive and therefore parallel computing is important [2,4,6,12,13]. In this section the parallel branch and bound algorithm with simplicial partitions and combination of Lipschitz bounds has been...
investigated. The results of different selection strategies have been compared. An
MPI version has been implemented using a parallel branch and bound template [1].
Static load balancing is used: tasks are initially distributed evenly (if possible) among
$p$ processors. If the initial number of simplices ($n!$) is less than the number of processors,
the simplices are subdivided until the number of processors is reached. Then the initial
simplices are distributed. After initialization, the processors work independently and
do not exchange any tasks generated later.
Parallel algorithm has been evaluated using standard criteria: speedup $s_p = t_1 / t_p$ and efficiency of parallelization $e_p = s_p / p$, where $t_p$ is time used by the algorithm implemented on $p$ processors. The averages $\bar{s}_p$ and $\bar{e}_p$ are shown in Table 6. For test problems of dimensionalities $n = 2$ and $n = 3$ the best average efficiency of parallelization with various numbers of processors $p$ is achieved when breadth first selection strategy is used. The efficiency of parallelization is very similar when best first and statistical selection strategies are used. The worst efficiency of parallelization for dimensionalities $n = 2$ and $n = 3$ is experienced when depth first selection strategy is used. For higher dimensionalities ($n \geq 4$) the average efficiency of parallelization is similar for all selection strategies. The efficiency of parallelization decreases less with the same number of processors for difficult (higher-dimensional) test problems compared with simpler test problems.

4 Conclusions

In this paper the speed and memory requirements of sequential branch and bound algorithm and efficiency of parallelization of parallel version of the algorithm has been investigated and compared for different selection strategies (best first, statistical, depth first and breadth first).

Optimization time is shorter when depth first and breadth first selection strategies are used. This is because of the time consuming heap structure required to prioritize candidates in the case of best first and statistical selection strategies. However the influence would be smaller for expensive objective functions which take longer to evaluate. The number of function evaluations required for the whole optimization are similar for all selection strategies, although depth first selection strategy requires the largest number of function evaluations. The number of function evaluations to locate the global solution is smallest when statistical selection strategy is used. Therefore this strategy is preferable when the solution time is limited. The maximal size of the candidate list varies much for different selection strategies. The smallest maximal size is when depth first selection strategy is used and it is up to $\sim 7,000$ times smaller than for other selection strategies. Therefore this selection strategy is preferable when memory is limiting. The maximal size of candidate list is up to $\sim 5$ times smaller when statistical selection strategy is used than when best first strategy is used. This explains why the optimization time is smaller when statistical selection strategy is used since the time required for insertion and deletion of candidates to/from heap structure depends on the number of elements in the heap.

The efficiency of parallelization is similar when best first, statistical and breadth first selection strategies are used. The efficiency of parallelization is worst when depth first selection strategy is used. The efficiency of parallelization is better for difficult test problems.

Acknowledgments This work was carried out under the HPC-EUROPA project (RII3-CT-2003-506079), with the support of the European Community – Research Infrastructure Action under the FP6 “Structuring the European Research Area” Programme. The research is partially supported by Lithuanian State Science and Studies Foundation within the project B-03/2007 “Global optimization of complex systems using high performance computing and grid technologies”.

Springer
References