Target Oriented Branch & Bound Method for Global Optimization

Volker Stix

Vienna University of Economics

Department of Information Business

Augasse 2–6

A-1090 Vienna / Austria

volker.stix@wu-wien.ac.at

May 2, 2001

Abstract

We introduce a very simple but efficient idea for branch & bound ($\mathcal{B}$&$\mathcal{B}$) algorithms in global optimization (GO). As input for our generic algorithm, we need an upper bound algorithm for the GO maximization problem and a branching rule. The latter reduces the problem into several smaller subproblems of the same type. The new $\mathcal{B}$&$\mathcal{B}$ approach delivers one global optimizer or, if stopped before finished, improved upper and lower bounds for the problem. Its main difference to commonly used $\mathcal{B}$&$\mathcal{B}$ techniques is its ability to approximate the problem from above and from below while traversing the problem tree. It needs no supplementary information about the system optimized and does not consume more time than classical $\mathcal{B}$&$\mathcal{B}$ techniques. Experimental results with the maximum clique problem illustrate the benefit of this new method.

Keywords — global optimization, branch and bound, maximum clique problem, standard quadratic optimization

1 Introduction and definitions

A general global optimization (GO) problem is of the following form:

$$\begin{aligned} f(\mathbf{x}) \quad &\rightarrow \max! \\ \mathbf{x} &\in \mathcal{M} \end{aligned} \tag{1}$$

where f is a real-valued function in $\mathbf{x}$, a vector which lies in the feasible set $\mathcal{M}$.

We let the *dimension* of a problem be the dimension of the vector $\mathbf{x}$. Our intention is to optimize globally, i.e. we are looking for a feasible vector $\hat{\mathbf{x}}$ such that $f(\hat{\mathbf{x}}) \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{M}$. Note that $\hat{\mathbf{x}}$ does not have to be unique. Already in very simple cases there may exist exponentially many (in the dimension of the problem) global optimizers. We refer e.g. to moon/mosers's classical result on cliques [17] for discrete GO problems, which forms a lower bound for possible global optimizers in quadratic problems (see, e.g. [8]) as an example for continuous GO problems. In this article we are only interested in one global optimizer.

Throughout the paper we assume that the following items are given:

1. **Objective function:** the function f of (1).
2. **Feasible set:** the feasible set $\mathcal{M}$ of (1) and methods for extracting points out of it.
3. **Upper bounds:** an upper bound value UB depending on (1), which guarantees that $UB_{f,\mathcal{M}} \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{M}$.
4. **Branching rule:** a rule which strictly decomposes a problem $p = (f, \mathcal{M})$ into i "smaller" subproblems $p_1, \ldots, p_i$. In this context, smaller means

either a strict smaller problem dimension or a strict smaller feasible set (i.e. $\mathcal{M}_1, \ldots, \mathcal{M}_i \subset \mathcal{M}$). In both cases the branching rule must guarantee that the global solution of the problem can be found in at least one of the generated subproblems. Reduction by dimension is more often used for discrete problem instances whereas partitioning the feasible set is more commonly used for continuous ones.

5. **A selection rule:** a rule which decides which constructed subproblem should be explored next. This rule can be a very simple one (depth-search-first or breadth-search-first) if there is little knowledge about the problem itself or it can be a more sophisticated heuristic depending on known structural patterns of the problem.

Note that the upper bounding algorithm need not necessarily be a very good one because our algorithm improves it during calculations. A nice trade-off, however, between the time needed to calculate UB and the quality of UB is assumed.

The required items listed above are quite natural for any $\mathcal{B}\&\mathcal{B}$ algorithm. In Section 2 such a classical $\mathcal{B}\&\mathcal{B}$ approach used for GO is described. In Section 3 we introduce a *target*, a value which should be reached at some $\mathbf{x} \in \mathcal{M}$. If this is not possible, then *target* acts as a new upper bound for the problem. The improvements of this target oriented $\mathcal{B}\&\mathcal{B}$ method are that the maximization problem is approximated not only from below by new better maximizers in $\mathcal{M}$ but also approximated from above by shrinking the upper bound. Therefore, this algorithm delivers also acceptable results even if the problem is too complex and the rendering process must be terminated before finishing. This is not unlikely for GO problems because already simple discrete problems like the maximum clique problem or continuous problems like standard quadratic optimization are known to be NP-hard to solve (see e.g. [11, 3]). In addition, a lot of problems

require the calculation of fewer subproblems using our target oriented $\mathcal{B}\&\mathcal{B}$ algorithm. This will be shown in Section 4. The basic idea of this approach is illustrated in Section 5 with a small example. Section 6 ends the paper with experimental results on the maximum clique problem, a discrete optimization problem.

2 Classical $\mathcal{B}\&\mathcal{B}$ methods

$\mathcal{B}\&\mathcal{B}$ methods are well known for a long time and often used in various field of continuous and discrete application domains. Horst/Tuy [14] describe the $\mathcal{B}\&\mathcal{B}$ method for continuous global optimization and criteria of convergence are developed which are necessary for infinite $\mathcal{B}\&\mathcal{B}$ procedures. A more compact overview for continuous problems can also be found in [13] or [12] with a lot of references within. They focus on the partition of the feasible set whereas the reduction in the dimension of the problem is more commonly applied to discrete GO problems (see e.g. [18, 9]) or [21] for a more implementation oriented point of view. Due to the finiteness of the feasible sets, discrete GO problems need no theory of convergence criteria. Continuous domain problems, however, can be as well reduced in subproblems by reducing its dimension as for example it is achieved for standard quadratic problems in [7]. Other recent works like [10, 15] concentrate on specific problem classes and their structural aspects. These are used to implement better selection heuristics in order to improve the efficiency of their algorithms compared with simple "black box" $\mathcal{B}\&\mathcal{B}$ approaches. In principle, however, a classical $\mathcal{B}\&\mathcal{B}$ approach as described below is used. It requires the inputs as we described already in Section 1 and the algorithm's outline looks like this:

Algorithm 1:

Input: The problem $p = (f, \mathcal{M})$.

A desired ϵ-precision.

Initialize: Set problem list $pl = \{p\}$.

Set as first maximizer any $\hat{\mathbf{x}} \in \mathcal{M}_p$.

Output: $\hat{\mathbf{x}}$ is one global maximizer of p.

1. Use the selection rule to remove the next problem $p \in pl$.
2. Use the branching rule to construct new subproblems $p_1, \ldots, p_i$ out of p.
3. For each p_i do
   (a) Calculate a lower bound l_i for p_i (e.g. by evaluating any feasible point in $\mathcal{M}_{p_i}$).
   (b) If $(l_i > f(\hat{\mathbf{x}}))$ then set $\hat{\mathbf{x}} = \mathbf{x}$, for $f(\mathbf{x}) = l_i$.
   (c) Calculate an upper bound u_i for p_i.
   (d) If $(u_i - l_i > \epsilon$ and $u_i > f(\hat{\mathbf{x}}))$ then set $pl = pl \cup p_i$.
4. Repeat from step 1 if $pl \neq \phi$.

Algorithm 1 starts with the first (main) problem together with a desired ϵ-precision as input. It chooses any $\hat{\mathbf{x}} \in \mathcal{M}$ as candidate for the global maximizer. In step 1 the next subproblem (which is the main problem in case of the first run) is removed from the list by the selection rule. This rule can be a simple one like LIFO or FIFO if pl is organized as a list, which semantically implements depth-search-first and breadth-search-first respectively. It can also be a more complicated heuristic regarding structural aspects of the problem. We do not care about this selection rule and assume it to be given. In order to converge to a global solution, however, for continuous $\mathcal{B}\&\mathcal{B}$ problems some additional requirements might be needed as discussed in the convergence section below. Step 2 decomposes the problem using the branching rule into "smaller"

instances. This is done either by reducing the problem's dimension or by partitioning the feasible set (or even both). Step 3 iterates through these newly generated subproblems and does the following: (a) it calculates a lower bound for the subproblem. This can be done by extracting one feasible point out of $\mathcal{M}_{p_i}$ and evaluating it under $f(\cdot)$. A more efficient alternative would be a (fast) local optimization procedure for p_i, if available. (b) it updates the best solution $\hat{\mathbf{x}}$ if necessary. Note that the vector itself rather than the value is updated. (c) it calculates an upper bound for the subproblem. For convergence reasons additional requirements might be added for the bounding procedure for continuous $\mathcal{B}\&\mathcal{B}$ problems as well (see below). (d) it decides whether it is necessary to explore p_i further. This is done (i) by testing if the desired precision is already reached and (ii) by testing whether the problem p_i is able to improve our so far best solution found by comparing u_i with $f(\hat{\mathbf{x}})$. Step 4 tests whether there are still subproblems left for exploration. After termination of Algorithm 1, $\hat{\mathbf{x}}$ is one global maximizer within an ϵ-precision. Its optimal value is $f(\hat{\mathbf{x}})$.

We can improve the initialization step as well as step (3a) by using not just any vector $\hat{\mathbf{x}}$ but a local optimizer of the respective problem, if it can easily be obtained.

2.1 Convergence

For discrete problems, ϵ can be set to zero and Algorithm 1 is finite because the branching rule guarantees a strict decomposition of the problems and thus the $\mathcal{B}\&\mathcal{B}$ tree is finite. For continuous problems some additional convergence criteria must be assumed as discussed in [14].

To summarize these it is required from the bounding operation to be *consistent*. This means that any infinitely decreasing sequence of successive refined

partitions $\mathcal{M}_q$ on $\mathcal{M}$ satisfies:

$$\lim_{q\to\infty}(l(\mathcal{M}_q) - u(\mathcal{M}_q)) = 0, \tag{2}$$

where $l(\cdot)$ and $u(\cdot)$ are lower and upper bounds resp. for the problem having $\mathcal{M}_q$ as feasible set. Stated in words, this means that whenever a problem's feasible region is refined small enough the lower bound of the refined region converges to the upper bound of that region. It is additionally required for consistency, that each problem being of interest can be refined further at each step.

With this consistency condition and an $\epsilon > 0$, Algorithm 1 is finite. This is because of (2), the required precision of step (3d) is achieved after finite many steps and therefore the $\mathcal{B}\&\mathcal{B}$ procedure is finite.

For $\epsilon = 0$ the selection rule has to be additionally *bound improving*. It must be ensured that the problem with the highest upper bound is refined infinitely often. If this is assured together with the consistency condition, then Algorithm 1 converges to the global solution. We refer to Horst/Tuy [14] for detailed discussion and proofs.

Note that we have distinguished three cases, where the discrete one together with any case with $\epsilon > 0$ is finite whenever the consistency condition holds. This condition can always be assumed for discrete problems, because at some level of decomposition the problems become trivial and hence upper and lower bounds coincide. The continuous case with full precision ($\epsilon = 0$) additionally requires the bound improving property for the selection rule and only convergence to the global solution can be guaranteed. Thus it can imply an infinite $\mathcal{B}\&\mathcal{B}$ procedure.

2.2 Examples for branching

The following two examples should illustrate the algorithm with two different branching rules. Figure 1 shows a simple one-dimensional function. Here the branching rule restricts the feasible region (interval) further. The lower bound

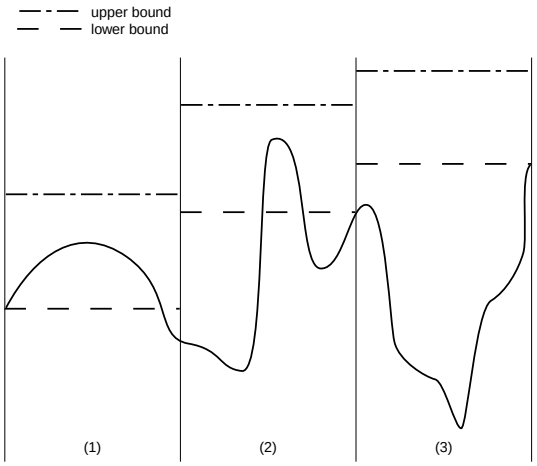

Figure 1: Continuous global optimization example.

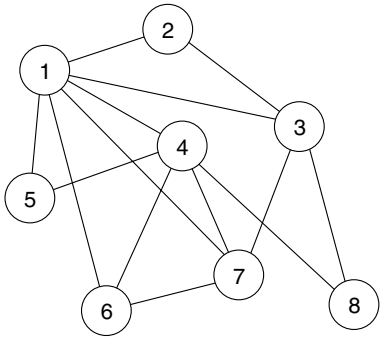

Figure 2: Discrete global optimization example.

is simply the higher function value at the boundaries of an interval. The upper bound is derived from the function though the details are not important here. We can see that partitioning interval (1) further is not efficient because its upper bound is lower than the lower bound of interval (3) which is the best solution up to that point. Therefore interval (1) can be discarded and only the remaining two intervals are subdivided again by the branching rule. A more realistic but not so illustrative continuous optimization problem, namely the standard quadratic problem, is discussed in the next section.

The second example is taken out of a discrete optimization domain. It is a well known problem in graph theory. We focus on undirected simple graphs, i.e. edges have no orientation and neither parallel edges nor loops are allowed (see Figure 2). A *clique* is a complete induced subgraph, i.e. a subset of nodes where all nodes are pairwise connected (i.e. $\{1, 2, 3\}$ is a clique). Searching for a clique with maximum cardinality is called the *maximum clique problem* (MCP). In our illustration, the nodes $\{1, 4, 6, 7\}$ form such a maximum clique. Many problems can be expressed in terms of the MCP and it is known to be NP-complete. For more detail about the MCP and its applications see the survey on MCP [5]. Obviously a branching rule for this problem should reduce the dimension of the problem instead of shrinking the feasible set. One idea as described in [1] is that a node can only form a maximum clique with its adjacent nodes. So in our example node 1 can only form a maximum clique with nodes $\{2, 3, 4, 5, 6, 7\}$. Therefore it is enough to search for the maximum clique in this new smaller problem and re-connect the solution to node 1 afterwards.

An example where a continuous optimization problem is decomposed by dimension and not through partitioning the feasible region can be found in [7]. This problem is connected to both, the MCP and standard quadratic optimization.

The given examples should serve only to motivate the idea of branching and are not further exploited here. In Section 6, however, we present promising results of experiments with the MCP and this new $\mathcal{B}\&\mathcal{B}$ algorithm.

3 Introducing a target

The improvement of the generic $\mathcal{B}\&\mathcal{B}$ algorithm presented in Section 2 for optimizing the problem $p = (f, \mathcal{M})$ is the introduction of a *target*. A target is a value which should be reached at least by a vector $\mathbf{x} \in \mathcal{M}$, i.e. $f(\mathbf{x}) \geq target$

should hold. Of course if $target$ is chosen too large, e.g. $target > UB_p$, then $f(\mathbf{x}) \geq target$ can never be satisfied. On the other hand if $target$ is chosen too small, e.g. $target = f(\mathbf{y}) = LB_p$ (for any $\mathbf{y} \in \mathcal{M}$), there is no challenge finding that particular $\mathbf{x}$ (simply choose $\mathbf{x} = \mathbf{y}$). If the target is chosen, however, to lie between LB_p and UB_p, e.g. $target = \frac{LB_p + UB_p}{2}$, it is unclear whether there is a $\mathbf{x} \in \mathcal{M}$ such that $f(\mathbf{x}) \geq target$. If such a vector $\mathbf{x}$ can be found, we have a new lower bound: $LB_p = f(\mathbf{x})$. If we fail finding such a vector $\mathbf{x}$ then a new upper bound is found: $UB_p = target$. In both cases the estimation of the global maximum of p is improved.

3.1 The algorithm

Before going into more detail we will show how this target oriented $\mathcal{B}\&\mathcal{B}$ method works and we will explain it afterwards.

Algorithm 2:

Input: The problem $p = (f, \mathcal{M})$.

A desired ϵ-precision.

Initialize: Set problem list $pl = \{p\}$.

Set remember list $rl = \phi$.

Set as first maximizer any $\hat{\mathbf{x}} \in \mathcal{M}_p$.

Calculate the global upper bound UB_g of the main problem p.

Set $target = \frac{f(\hat{\mathbf{x}}) + UB_g}{2}$.

Output: $\hat{\mathbf{x}}$ is one global maximizer of p.

1. Use the selection rule to remove the next problem $p \in pl$.
2. Use the branching rule to construct new subproblems $p_1, \ldots, p_i$ out of p.
3. For each p_i do

(a) Calculate a lower bound l_i for p_i (e.g. by evaluating any feasible point in $\mathcal{M}_{p_i}$).

(b) If $(l_i > f(\hat{\mathbf{x}}))$ then set $\hat{\mathbf{x}} = \mathbf{x}$, for $f(\mathbf{x}) = l_i$ and set $target = \frac{f(\hat{\mathbf{x}})+UB_g}{2}$ if $(f(\hat{\mathbf{x}}) \geq target)$.

(c) Calculate an upper bound u_i for p_i.

(d) If $(u_i - l_i > \epsilon)$ then

If $(u_i \geq target)$ then

set $pl = pl \cup p_i$.

else If $(u_i \geq f(\hat{\mathbf{x}}))$ then

set $rl = rl \cup p_i$.

4. Repeat from step 1 if $pl \neq \phi$.

5. Set $pl = rl$, $UB_g = target$, $target = \frac{f(\hat{\mathbf{x}})+UB_g}{2}$.

6. If $(UB_g - f(\hat{\mathbf{x}}) > \epsilon$ and $pl \neq \phi)$ repeat from step 1.

We will call the completion of the inner loop (step 1–4) a *run* because it is structurally similar with one "run" through Algorithm 1. Thus step 6 can force the algorithm to *re-run* from step 1.

Lemma 1 *With the assumptions made in Section 1, and the additional requirements for continuous problems as discussed in Section 2.1, Algorithm 2*

1. *is finite for discrete problems and continuous problems having* $\epsilon > 0$.

2. *converges for continuous problems having* $\epsilon = 0$.

Proof Ad 1). For these problems the inner loop (step 1–4) constructs in step 3d) only finite many problems for the problem list pl as well as for the remember list rl thus the inner loop is finite. Note that for continuous problems, the bounding processes is required to be consistent (Section 2.1). The outer loop (step 1 to 6) is finite because in step 5 the gap $\delta := target-$

$f(\hat{\mathbf{x}})$ is halved. $f(\hat{\mathbf{x}})$ is monotonic increasing which can only decrease δ. If the gap becomes negative (step 3b), it is reset to at least one halve of the previous gap and so on. After finite many re-runs (step 6) the ϵ-precision is reached. The algorithm requires at most $\log_2((UB_f - f(\mathbf{x}))/\epsilon)$ re-runs of the outer loop. Ad 2). For these problems the selection rule is additionally required to be bound improving (Section 2.1). This is possible, because the problem with the highest upper bound is always in the problem list *pl* (step 3d), which enables the selection rule to refine this problem infinitely often. Thus convergence can be guaranteed.

Lemma 2 *Increasing the target variable (i.e. the threshold for the bounding process) during runtime (step 1), leaves the system consistent, meaning that all branches rejected earlier would be rejected with this new target value as well.*

Proof Branches have been rejected because the upper bound of that branch was smaller than target at that time. Increasing the target variable maintains this order.

Lemma 3 *The variable target forms an upper bound of the main problem once all nodes in pl are visited (after step 4).*

Proof All branches with upper bounds smaller than *target* are discarded or put in *rl*. Because of Lemma 2, this rejection holds even if *target* is increased in step 3b. All other branches are exploited further until none has an upper bound exceeding *target*, which proves *target* to be the new upper bound.

Theorem 4 *Algorithm 2 delivers the global optimum together with one optimizer of problem (1) within precision ϵ with assumptions made in Section 1 and requirements discussed in Section 2.1 in finite time or at least converges to it for $\epsilon = 0$.*

Proof Lemma 1 covers the finite and convergence property. It finds the global optimum because of the breaking condition in step 6 and by Lemma 3.

Here follows how Algorithm 2 works. Algorithm 2 is very similar to Algorithm 1 except for minor changes. The most salient ones are in steps 3d and steps 5–6. In step 3d, besides precision, successive subproblems are only investigated further if their upper bound is not smaller than $target$ (instead of $f(\hat{\mathbf{x}})$ as in Algorithm 1). This would imply that we have already found an optimizer $\tilde{\mathbf{x}}$ where $f(\tilde{\mathbf{x}}) = target$. Therefore all branches where the upper bound can not cope with these "harder" requirements are cut back. Consequently, under normal conditions, much fewer subproblems are calculated during recursion. In step 5, by Lemma 3, the newly gained information is stored (UB_g) and the target value is re-initialized. Because the target was set so high (there never was such a $\tilde{\mathbf{x}}$ as implied before), we might have discarded efficient sub problems. Therefore we have to look at these remembered problems in rl ($pl = rl$) with the newly gained information (step 6). If on the other hand $UB_g - f(\hat{\mathbf{x}}) \leq \epsilon$, we found the global maximizer to be $\hat{\mathbf{x}}$.

We should examine step 2b as well where the statement $target = \frac{f(\hat{\mathbf{x}})+UB_g}{2}$ carries over dynamically the idea of the target once there is a $\hat{\mathbf{x}}$ with $f(\hat{\mathbf{x}}) \geq target$. Again the goal is set higher than necessary. Because $target$ is only monotonically increasing there is no problem of inconsistency by changing the target value at run time as we showed in Lemma 2.

3.2 A continuous example

The standard quadratic problem (StQP) was mentioned already several times before. It is the optimization of a quadratic form over the standard simplex, i.e.

$$\begin{aligned} \mathbf{x}^t Q \mathbf{x} \quad &\rightarrow \max! \\ \mathbf{x} \in \Delta_n, \end{aligned} \tag{3}$$

where $\Delta_n = \{\mathbf{x} : x_i > 0, i = 1, \ldots, n, \sum_{i=1}^{n} x_i = 1\}$ is the standard simplex and n the dimension of the problem. Due to the structure of Δ_n the linear term of the quadratic form can be coded into the matrix Q. Any quadratic optimization problem can be solved by solving some StQPs. They are very commonly used and appear in a lot of applications. This is not at least because of the fact that many combinatorial problems or integer optimization can be reduced to StQPs as well. This offers the advantage to solve discrete problems with discrete and continuous optimization tools. Please see Bomze [3] for a discussion of these problems.

Besides the function and the feasible set given in (3) an upper bound, a branching rule and a selection rule is required as input. Additionally a local optimizer for lower bounds is suggested. Bomze discusses in [4] all these aspects for StQPs. Some of them are mentioned here. The upper bound can be achieved i.e. by d.c. decomposition where the quadratic form $Q = Q_+ + Q_-$ consisting of a convex (Q_+) and concave (Q_-) part is represented as the difference of two convex functions $Q = Q_+ - (-Q_-)$. Note that $-Q_-$ is convex. With this decomposition an upper bound can be achieved by maximizing Q_+ which is a convex maximization problem and minimizing $-Q_-$ which is concave maximization problem. Both problems are well known and discussed e.g. in [13, 14, 4]. The branching process can be done by partitioning the feasible set via simplicial decomposition [13, 14] or through decomposition by dimension as described e.g. in [7, 4]. The selection rule should be adapted to heuristics of the underlying problem modelled by the StQP. As a local optimization procedure a very robust dynamical system, the *replicator dynamic*, is suggested, which delivers good local optimizers within reasonable time [6, 7, 19]. With these inputs, which are required by the classical $\mathcal{B}\&\mathcal{B}$ algorithm as well, the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm can be used.

4 The runtime complexity of the new approach

It is hard to compute different $\mathcal{B}\&\mathcal{B}$ systems among each other theoretically, because they are highly dependent on problem instances. There are easy ones and there are hard ones for almost every kind of algorithm (they can be constructed). In our experiments we observed, however, high gains in performance and quality. The latter one is given through the improvement property of the upper bounds and was discussed in the previous section. The improvement in performance can be explained as follows.

4.1 Comparison of the two algorithms

In the previous section it may appear that the number of re-runs (step 6) of Algorithm 2 serve to lengthen calculation time when compared to the classical $\mathcal{B}\&\mathcal{B}$ Algorithm 1. This can be negated for almost all the time, for problem instances being of interest.

Suppose that the whole problem tree were given in advance. This would be the same tree independently of the working algorithm because we are using the same branching rule. Now color only those nodes of the tree which the classical $\mathcal{B}\&\mathcal{B}$ algorithm will visit, depending on its bounding capabilities (i.e. never visit nodes (and sub-nodes) where the upper bound of this node is smaller than my current best $f(\hat{\mathbf{x}})$). It should be clear also, that none of the discarded nodes will be visited by the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm since it has an even more rigorous cutting rule (i.e. $target$ is never smaller than $f(\hat{\mathbf{x}})$). So at most all colored nodes of the problem tree will be searched by the new algorithm. It is, however, rather unlikely that it will render all of those, because due to its high $target$ value, the algorithm "forces" the improvements of $f(\hat{\mathbf{x}})$ to be rather large. This can be thought of as the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm first searches some of the colored nodes fast and rough in order to find a very large improvement.

Should it fail, it searches then some of the remaining colored nodes, refining its precision with each re-run. In the worst case scenario, it would search all of the colored nodes to find the global optimum like the classical $\mathcal{B}\&\mathcal{B}$ algorithm does. Because the coloring is a dynamical process and highly depends on the problem itself the arguments above hold only most of the time.

It could be argued that the order of the colored visited nodes differs between these two algorithms. Therefore it might happen that the classical $\mathcal{B}\&\mathcal{B}$ method finds the real global optimizer $\hat{\mathbf{x}}$ very soon and therefore renders the rest of the tree, which would only be necessary to prove that $\hat{\mathbf{x}}$ is the global maximizer, in the most optimal way. This can happen though it is very unlikely given the complex global optimization problems being of interest. This is because finding the global optimizer very early would imply that we need the $\mathcal{B}\&\mathcal{B}$ algorithm almost only for validation whether a (local) optimum $f(\hat{\mathbf{x}})$ found at the beginning is already the global one. It was seen in experiments with a stochastic $\mathcal{B}\&\mathcal{B}$ problem [20] that whenever the problem was structured that the global optimizer was found easy and fast, the two algorithms almost performed the same. For harder problems, where the global optimizer could not been extracted so easily and fast and was "hidden" behind smaller inefficient local solutions, the target oriented approach always was faster, at least by 30%. Even better results can be found in Section 6.

Target oriented $\mathcal{B}\&\mathcal{B}$ does not imply any selection rule. Thus the selection rule used for the classical $\mathcal{B}\&\mathcal{B}$ approach can be used as well for the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm. This new approach clusters the problem tree in much larger segments. Within these the same selection rules can be applied yielding the same effect on its assumed gains. This makes this approach very interesting because it can be simply applied without further knowledge of the problems.

4.2 Better results during runtime

In many problems of global optimization, the problem tree is so huge that it can never be rendered in a reasonable time. Such NP-complete problems appear very soon and some of them are too hard to solve with the current computer power and state of research. We will show some of these hard problems in Section 6 as well. Such problems will always exist as long as $P \neq NP$. For these problems the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm can also support our effort in finding the global optimum better than the classical one. Algorithm 2 improves the gap between upper bound for the problem and $f(\hat{\mathbf{x}})$ after each re-run in a geometric way. Because the number of nodes rendered in the first run are much fewer compared to those of the classical $\mathcal{B}\&\mathcal{B}$ method, it is much more likely that Algorithm 2 succeeds in finishing its first run. At this point we have at least an improved upper bound and very likely also an improved lower bound. Then the first re-run starts (step 6) to refine our results. If it also succeeds in finishing its task again a better upper bound is achieved. With each re-run the number of problems to be searched will increase and at some re-run level it might again be not possible to finish all the problems of that particular tree within reasonable time. Until then, however, we have improved upper and lower bounds.

A problem can be so complex that even the first run of the target oriented $\mathcal{B}\&\mathcal{B}$ approach does not succeed. In these cases we can attempt to gain more information during runtime with a small meta-programm as shown below. We will set T, the maximum amount of time, which we are willing to spend in solving a problem.

Algorithm 3:

Input: Problem $p = (f, \mathcal{M})$.

precision ϵ.

Maximum time T.

Initialize: Initialize Algorithm 2.

Set t=$\frac{T}{\log_2((UB_g - f(\hat{\mathbf{x}}))/\epsilon)}$.

Output: $\hat{\mathbf{x}}$ is best maximizer and UB_f is best upper bound for p.

1. Start with step 1 of Algorithm 2 until step 5 (without re-runs).
2. If it is not finished with this after time t, STOP it, keep the best lower bound $\hat{\mathbf{x}}$ and set new $target = \frac{target+UB_g}{2}$. If it succeeds, however, set $UB_g = target$ as the new upper bound for the problem.
3. Repeat from step 1 if time T is not exceeded.

The algorithm above is just a simple outline and should provide the basic idea. We know that we have at most $\log_2((UB_f - f(\hat{\mathbf{x}}))/\epsilon)$ re-runs therefore we should not spend more than t time units for one run. If we are not able to prove $target$ to be an upper bound within t time units, we set $target$ higher and re-run for another t time units. It is now more likely than before, that this time the first run succeeds because the target is higher. At the end we either finished the whole problem as described earlier (very unlikely) or we have proved some new upper bound and increased the lower bound (very likely) or we improved neither the upper nor the lower bounds (again very unlikely). The new approach, however, allows us to somehow partition the maximum available time in a reasonable fashion in order to achieve more than we had in the beginning.

5 The benefit illustrated with a simple example

The following example tries to outline the benefits of this target oriented $\mathcal{B}\&\mathcal{B}$ method. Though the example might appear artificial, this is necessary given that in normal applications, the problem tree is too large to be used as an elucidatory example.

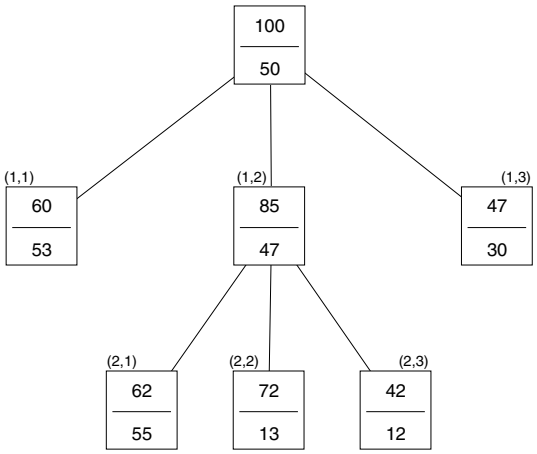

Figure 3: First run through the tree. $UB_f = 100$, $target = 75$.

For simplicity, avoiding lots of decimal points, we chose a discrete optimization example, therefore $\epsilon = 1$. The upper numbers inside the nodes are the upper bounds of this sub problem and the lower numbers are the local solutions (lower bounds) inside these problems.

Figure 3 shows the first run through the tree. The upper bound UB_g for the main problem is 100 and $f(\hat{\mathbf{x}}) = 50$ for some arbitrary chosen $\hat{\mathbf{x}}$ and thus $target = 75$. The branching rule then generates the problems (1,1),(1,2) and (1,3) and their local estimators (lower bounds) and upper bounds. A better optimizer $\hat{\mathbf{x}}$ is found in problem (1,1) with $f(\hat{\mathbf{x}}) = 53$ but the upper bound of this problem is smaller than $target$, therefore this problem is rejected this time (unlike the classical $\mathcal{B}\&\mathcal{B}$ method) but remembered on a list. Problem (1,3)'s upper bound does not exceed $f(\hat{\mathbf{x}})$ and is therefore discarded (like in the classical $\mathcal{B}\&\mathcal{B}$ method). Problem (1,2)'s upper bound exceeds $target$ and therefore new subproblems and their values are constructed. A new maximizer is found in (2,1) with $f(\hat{\mathbf{x}}) = 55$. Again the upper bound is too small as it is in problem (2,2), therefore both are rejected this time but remembered whereas problem (2,3) is discarded permanently the same as problem (1,3). At the end of this run

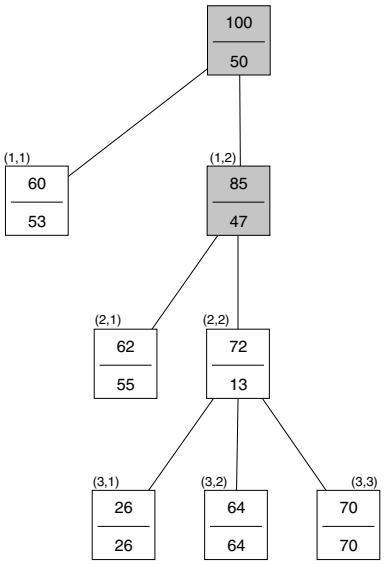

Figure 4: Second run through the tree.$UB_f = 75$, $target = 65$.

(step 5) we have $f(\hat{\mathbf{x}}) = 55$ and a new proven upper bound of 75. Additionally we know that only problems (1,1),(2,1) and (2,2) should be considered for the next run ($pl = rl$). Our new target now is $65 = (75 + 55)/2$.

Figure 4 shows the second run through the tree. The new upper bound now is 75, which is not considered in the main node and node (1,2) because these nodes are rendered and are no longer visited. They appear in the illustration simply for better orientation. We start with problem (1,1) and see that this problem's upper bound is still too low. We reject and remember it. The same result is valid for node (2,1). Node (2,2), however, constructs new subproblems, which are this time easy enough to be rendered globally (i.e. upper and lower bounds coincide). We improve the maximum to 64 in problem (3,2) and to 70 in problem (3,3), which resets the target value to $72,5 = (70 + 75)/2$ (step 3b)and after that this run is finished. We proved that 72,5 is an upper bound and found $f(\hat{\mathbf{x}}) = 70$ and know that we now only have to look at problems (1,1) and (2,1). The third run now is trivial with no computation and ends with the proof that

the global optimum is 70.

In contrast to the classical $\mathcal{B}\&\mathcal{B}$ method not only problems (1,3) and (2,3) were never expanded, but also problems (1,1) and (2,1). The classical method might got stuck in the left-most branch starting with problem (1,1). Once again, this is an artificial problem, constructed to illustrate the main points.

6 Experimental results of hard problems

We made some experiments with the MCP in graph theory as already presented in Section 2. The classical $\mathcal{B}\&\mathcal{B}$ method and the target oriented algorithm was applied to some of these problems. As a branching rule the basic idea of Babel [1] is chosen. The selection rule, however, is coded without any heuristic ordering of the nodes. The upper bound is given through a valid coloring of the graph which delivers an upper bound for the MCP [2, 16]. The coloring algorithm is very simple and no attempt was made in optimizing it. Lower bounds are extracted during the coloring process. The graphs are taken from the DIMACS database [16] and are known to be difficult to solve. Because the graphs are artificially constructed, in most cases the real maximum clique size is known and printed in column "real maximum" in Table 1. The column "classical method" shows the best found maximum using the classical branch and bound algorithm. The next two columns show lower and upper bounds of the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm and in the last two columns the time (in seconds) required for the classical method and the target oriented $\mathcal{B}\&\mathcal{B}$ algorithm can be found. Here ∞ stand for a kill of the algorithm after 6 hours. Graphs in bold face are problems, where the target oriented $\mathcal{B}\&\mathcal{B}$ method found not only the maximum but also proved that the found maximum is the global one ($UB = LB$). It is interesting that problems seems to be solved either "immediately" or "never". This kind of behavior was also found in [20] with a stochastic optimization problem using

			real	classical	target		time	
Graph	Size	Density	max.	method	LB	UB	class	target
brock200_1	200	0.745	21	21	21	21	697	349
brock200_2	200	0.496	12	12	12	12	1	1
brock200_3	200	0.605	15	13	15	15	∞	5
brock200_4	200	0.658	17	16	17	17	∞	18
brock400_1	400	0.748	27	20	25	33	∞	∞
brock400_2	400	0.749	29	23	24	41	∞	∞
brock400_3	400	0.748	31	23	24	42	∞	∞
brock400_4	400	0.749	33	23	24	41	∞	∞
p_hat500-1	500	0.253	9	9	9	9	3	2
p_hat500-2	500	0.505	36	36	36	41	∞	∞
p_hat500-3	500	0.752	≥ 49	47	48	68	∞	∞
p_hat700-1	700	0.249	11	9	11	11	∞	7
p_hat700-2	700	0.497	44	42	44	60	∞	∞
p_hat700-3	700	0.748	≥ 62	59	59	114	∞	∞
san200_0.7_1	200	0.700	30	17	30	30	∞	1
san200_0.7_2	200	0.700	18	14	18	18	∞	6
san200_0.9_1	200	0.900	70	48	70	70	∞	23
san200_0.9_2	200	0.900	60	40	60	60	∞	328
san200_0.9_3	200	0.900	44	35	41	46	∞	∞
san400_0.5_1	400	0.500	13	12	13	13	∞	4
san400_0.7_1	400	0.700	40	21	40	40	∞	122
san400_0.7_2	400	0.700	30	17	30	30	∞	566
san400_0.7_3	400	0.700	22	15	17	25	∞	∞
san400_0.9_1	400	0.900	100	56	99	108	∞	∞

Table 1: Performance of DIMACS Graphs using target oriented $\mathcal{B}\&\mathcal{B}$ Algorithm

target oriented $\mathcal{B}\&\mathcal{B}$. In our opinion this is simply the fact that some NP-complete problems are easier but there exist for each problem class a threshold beyond that the problems become very fast intractable for all algorithms (unless $P = NP$).

It can be noted that especially for the Sanchis graph family, the target oriented $\mathcal{B}\&\mathcal{B}$ approach was very successful. This is because in this family there are many local maximizers (maximal cliques) with cardinalities at about half the real maximum clique. The target oriented $\mathcal{B}\&\mathcal{B}$ approach forces the computation to "jump" over these many inefficient small local maxima as discussed in Section 4. In some cases the target oriented approach was not able to render the problems though it did succeed in approximating them nicely from above and below.

7 Conclusions

We may conclude that the target oriented $\mathcal{B}\&\mathcal{B}$ method is more efficient than the classical one. It is independent of the problem, and improves upper and lower bounds better and faster than the classical method and it is not much more difficult to implement. The drawback is the remembering of the rejected nodes, which consumes memory resources.

In principle, *target* does not have to be the arithmetic mean between upper and lower bound. Any better estimate depending on the problem could do a better job. Nevertheless target must be monotonic increasing in order to preserve consistency during re-runs. The upper and lower bounds can be thought of as hard bounds, and the target as an estimation of the global optimum inside the problem. If the estimate is close to the real optimum, the algorithm can prove much more quickly that the optimum has been found.

References

[1] L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38, 1994.

[2] B. Bollobás. *Modern graph theory*. Springer, New York, 1998.

[3] I. M. Bomze. On standard quadratic optimization problems. *Journal of Global Optimization*, 13:369–387, 1998.

[4] I. M. Bomze. Branch-and-bound approaches to standard quadratic optimization problems. Technical Report, to appear in JOGO, TR-ISDS 2000-14, Department of Statistics and Decision Support Systems, University of Vienna, 2000.

[5] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, volume Suppl. Vol. A:4, Boston, MA, 1999. Kluwer Academic Publishers.

[6] I. M. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11(6):1228–1241, 1999.

[7] I. M. Bomze and V. Stix. Genetic engineering via negative fitness: Evolutionary dynamics for global optimization. *Annals of Oper. Res.*, 89, 1999.

[8] M. Broom, C. Cannings, and G. T. Vickers. On the number of local maxima of a constrained quadratic form. *Proc. R. Soc. Lond.*, 443:573–584, 1993.

[9] W. Cook. *Combinatorial Optimization*. Wiley, New york, 1998.

[10] M. Djerdjour and K. Rekab. A branch and bound algorithm for designing reliable systems at a minimum cost. *Applied Mathematics and Computation*, 118:247–259, 2001.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide the the Theory of NP-Completeness*. Freemann, New York, 1995.

[12] R. Horst and P. M. Pardalos. *Handbook of global optimization*. Kluwer, Dordrecht, 1995.

[13] R. Horst, P. M. Pardalos, and V. N. Thoai. *Introduction to global optimzation*. Kluwer, Dordrecht, 1995.

[14] R. Horst and H. Tuy. *Global Optimization*. Springer, Heidelberg, 3rd edition, 1996.

[15] S. Jain. Branch and bound on the network model. *Theoretical Computer Science*, 255:107–123, 2001.

[16] D. S. Johnson and M. A. Tricks (editors). Cliques, coloring and satisfiability: Second dimacs implementation challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, American Mathematical Society, Procidence, 1996.

[17] J. W. Moon and L. Moser. On cliques in graphs. *Isr. J. Math.*, 3:23–28, 1965.

[18] R. G. Parker. *Discrete Optimization*. Academic Press, Boston, 1988.

[19] V. Stix. The maximum clique problem—a fresh approach: Empirical evidence and implementation in C++. *Thesis submitted for diploma, University of Vienna.*, 1997.

[20] V. Stix. Stochastic branch & bound applying target oriented branch & bound method to scenario tree optimization. Refereed conference paper at stochastic programming 2001, Department of Information Business, Vienna University of Economics, 2001.

[21] M. A. Weiss. *Data structures & algorithm analysis in C++.* Addison-Wesley, Reading, 1999.