Comparison of Strategies for Solving Global Optimization Problems Using Speculation and Interval Computations

Angel F. Garcia Contreras and Martine Ceberio
Department of Computer Science
The University of Texas at El Paso, USA
Email: afgarciacontreras@miners.utep.edu, mceberio@utep.edu

Abstract—Many real-life situations require that a match between quantities, behaviors, etc. be found. That is the case, for instance, when scientists try to find a fit between two sets of data, or a set of observations and a given model. Often such situations require that the minimum (or maximum) of a computed difference be found. These situations can be modeled as optimization problems. There exist multiple flavors of optimization problems: constrained and unconstrained (whether we are looking for the minimum – or maximum – of a function over the entire search space or only within the subspace of elements that satisfy some given constraints); local and global (whether we are looking solutions for the minimum within a neighborhood or among the whole search space); continuous, discrete, and mixed (whether the parameters of the problem at hand take their values all in discrete domains, all in continuous domains, or in a mix of these). In this article, we focus on continuous unconstrained global optimization and algorithms to solve such problems. Without loss of generality, we will discuss minimization.

There exist many algorithms to address such problems. Most are based on interval computations for they provide a way to conduct a fully covering search in continuous domains where enumeration of alternatives is impossible. In this article, we propose to look at a specific type of algorithm: known as speculation, which consists in betting on which value is going to be the minimum we are looking for. More specifically, we propose to improve our speculative approach using different strategies. We present and discuss the results of a series of experiments comparing the performance of the speculative algorithm with the proposed strategies.

I. INTRODUCTION

Many real-life situations require that a match between quantities, behaviors, etc. be found. That is the case, for instance, when scientists try to find a fit between two sets of data, or a set of observations and a given model. Often such situations require that the minimum (or maximum) of a computed difference be found. These situations can be modeled as optimization problems. There exist multiple flavors of optimization problems: Optimization can be constrained or unconstrained whether we are looking for the minimum – or maximum – of a function over the entire search space or only within the subspace of elements that satisfy some given constraints. Optimization can be local or global whether we are looking solutions for the minimum within a neighborhood or among the whole search space. Optimization can be continuous, discrete, or mixed (whether the parameters of the problem at hand take their values all in discrete domains, all in continuous domains, or in a mix of these). In this article, we focus on continuous unconstrained global optimization and algorithms to solve such problems. Without loss of generality, we will discuss minimization only.

There exist many algorithms to address such problems. Unlike local algorithms that can very quickly identify a reasonably good solution to an optimization problem (but without guarantee that it would be even close to the best solution), global search algorithms guarantee that their result is a global optimum, trading speed for rigor and reliability. There are usually two approaches to global search: analytical and exhaustive. Analytical methods require that the problem be with certain conditions or properties that can guarantee optimality of the solution. Exhaustive search usually follows a divide-and-conquer approach. The most well-known type of exhaustive search algorithm is Branch & Bound. The idea behind Branch & Bound is to divide the original search area into progressively smaller sub-spaces and at each step of the way, to evaluate the likelihood that the sub-space at hand can be / contain a solution. Most Branch & Bound-based continuous optimization algorithms rely on interval computations for they provide a way to conduct a fully covering search in continuous domains where enumeration of alternatives is impossible.

We recall existing Branch & Bound algorithms and solvers. BARON [1] is a commercial award-winning mixed-integer nonlinear optimization solver that uses interval arithmetic, convexification and relaxation to solve non-convex optimization problem. While computationally efficient, BARON uses non-rigorous relaxations and cannot guarantee the solutions it finds are global optima. GlobSol [2] is an open-source solver for solving unconstrained / constrained global optimization and nonlinear systems of equations that uses rigorous interval arithmetic, guaranteeing the global quality of its reported solutions. ALIAS-C++ [3] is a library of algorithms for solving optimization and systems of equations. While not exactly an optimization solver, it implements a parameterized branching algorithm and consistency algorithms with the Profil/BIAS [4] fast interval library. IbexOpt [5] is a module of the Ibex constraint processing library. This module uses Branch & Bound with the interval techniques of Ibex to find guaranteed solutions to global non-convex optimization problems.
introduced by interval arithmetic, and improving execution
time through new sub-problem generation and selection heuris-
tics.

In previous work, we presented a speculative optimization
method [6], [7] that includes the objective as an additional
branching dimension. It has similarities with the graph subdivi-
sion methods presented by Shary [8], which also introduces
the range of the objective as a branching dimension. Speculative
optimization places a greater priority on dividing the objective
over the domain. In this article, we present strategies to
improve this speculative algorithm. We present and analyze
the results of experiments that compare the execution time for
various configurations of the speculative algorithm as well as
a standard Branch & Prune algorithm.

II. BACKGROUND

A. Optimization

Unconstrained continuous global optimization problems are
defined as follows:

$$\min_{x \in \mathbb{R}} f(x)$$  \hspace{1cm} (1)

where \( f : \mathbb{R} \to \mathbb{R} \) is called the objective function. In
this work, we focus on minimization without loss of generality
since maximizing is equivalent to minimizing \(-f\). Function \( f \)
is continuous, potentially non-convex functions. Solving such
an optimization problem consists in identifying \( x^* \), a value
of \( f \)'s variable \( x \), such that \( \forall x \in \mathbb{R}, f(x^*) \leq f(x) \).

Search algorithms used to find solutions of optimization
problems are divided in two categories: local search and
global search. Local search usually starts with an initial guess
\( x_0 \in [x] \). Based on this guess, the algorithm iteratively
generates new points \( x_i \) s.t. \( f(x_i) \leq f(x_{i-1}) \), until there
is no improvement or \( f(x_i) = f(x_{i-1}) \). Local algorithms
converge quickly to the solution nearest to the initial point,
but may struggle (or even not converge) if the initial point is
far from a solution. To find better solutions, some algorithms
restart their search from different initial points, recording the
best solution. They find a local solution that has no guarantee
of being the best solution in the domain, only the best in
a neighborhood of \( x_0 \). However, certain situations require
finding the guaranteed best value of the objective. Global
search methods guarantee finding \( x^* \) that yields the best
objective value. These algorithms consider the entire domain
of \( x \). The strategy consists in iteratively reducing the space
of the search, generating smaller subdomains. The search ends
when sufficient precision has been reached. Such algorithms
rely on interval computations to be able to conduct such
search: in what follows, we provide some background about
intervals and domain reduction, as well as how they integrate
into a global search algorithm.

B. Interval Computations

An interval \( x_i = [x_{i1}, x_{i2}] \) is the set of all real numbers \( x \)
such that \( x_1 \leq x \leq x_2 \). We can \( I \) the set of all such intervals.
More generally, we call a box \( X \) the Cartesian product of
intervals: e.g., \( X = x_1 \times x_2 \times \ldots \times x_n \in I^n \).

Interval arithmetic extends real arithmetic to intervals. All
arithmetic operators \( \in \{+, -, \times, \div\} \) are redefined on
intervals in a way that ensures that:

$$\{ x \triangleright y \mid x, y \in \mathbb{R}, y \neq 0 \} \subseteq x \triangleright y$$

Similarly, all real functions \( f \) can be “extended” to intervals,
as \( f \), by systematically extending each of the operators of their
symbolic expression to intervals. As a result, we have that:

$$\forall X \in I^n, \{ f(x) \mid x \in X \} \subseteq f(X)$$

More details about interval analysis can be found at [9].

As we can observe on Fig. 1, interval computations allow
us to enclose the range of function \( f \) over interval \( x \), but
the enclosure is not exact. This is caused by the so-called
dependency problem [9], which causes interval evaluations
to often overestimate the computed quantities. Even simple
function such as \( f(x) = x - x \) produce an overestimation
when extended to intervals. Indeed if we extend \( f \) to intervals
and evaluate it on \( x = [1, 2] \), then we obtain \( f(x) = [1, 1] \).

Dealing with overestimation to yield more exact interval
evaluations is the subject of many pieces of research work: for
instance, symbolic transformations that reduce the number of
times a variable appears in the expression [10], reformulations
of the evaluation such as centered form [9], and evaluation
of the function using the monotonicity of each term of the
expression [11] However, this is not the focus of this article
and the only property that matters here is that we can rely
on the fact that intervals provide a correct enclosure of the
quantities being computed (in particular in our case, the range
of objective functions).

C. Contractors

Constraints limit the values parameters can take in a given
domain. Although in this work we focus on unconstrained
optimization, interval-based solving techniques usually model
problems as constraints to reduce the size of the (box) do-
mains: they combine contraction of the domains via contractors
with domain splitting. Figure 2 illustrates the application
of a contractor to a domain. Contractors \( C \) are iterative
operators that attempt to reduce the domain at hand based
on each individual constraint of the problem, shrinking the
respective domains of each of the parameters of the problem
to exclude values that violate the constraints. As a result, when
a contractor is applied to a domain \( x \), we have: \( x = C(x) \cap x \).
The desired outcome of this is that either $x$ be reduced or discarded (when $C(x) \cap x = \emptyset$).

In previous work [7], [13], we presented a speculation-based B&P algorithm for global optimization with a new bisection rule. Our speculative algorithm:

1) First evaluates the objective function using interval arithmetic to obtain a range that contains all possible values for the objective, $f_1 = [L, U]$;

HC4 [12] is an example of a popular interval constraint-based contractor, which we use in our work. There exist many other contractors. For instance, the Interval Newton method [9] solves equations using intervals: $F(x) = 0$ with $F : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$. The real-number version of the Newton method starts with an initial point of the search space and iteratively improves the evaluated point until it converges to a solution (if it converges). The interval version of this algorithm starts with an initial interval box $D$ in which the fixed-point solution is sought. The algorithm iteratively shrinks this box domain while still containing the zero of $F$ if it is there. The Interval Newton operator as a contractor $N$ works as:

$$D_{k+1} = N(D_k) \cap D_k,$$

starting with an initial box $D$.

A variation of the Interval Newton method is the Krawczyk method [9]. Just like Newton, this algorithm solves systems of non-linear equations using intervals. The core element of this method is the Krawczyk operator, defined as:

$$K(x) = y - Yf(y) + \{I - YF'(x)\}(X - y),$$

where $Y$ is a nonsingular real matrix approximating the inverse of the real Jacobian matrix $F'(m(x))$ with elements $F'(m(x))_{ij} = \partial f_i(x)/\partial x_j$ at $x = m(x)$, $y$ is a real vector contained in the interval vector $x$, and $m(x)$ is the vector containing the midpoints of box $x$. If $K(x) \subseteq x$, then both $x$ and $K(x)$ contain a solution to the system of equations. The Krawczyk method is the application of $K(x)$ to the definition of Interval Newton:

$$x_{k+1} = K(x_k) \cap x_k,$$

where $x_0$ is a starting interval box s.t. $K(x_0) \subseteq x_0$, and the solution box is $x^* = x_k$ s.t. $x_{k+1} \approx x_k$ [9]. In the work we present in this article, we use the Krawczyk operator.

**D. Interval Branch & Bound, Branch & Prune**

Interval Branch & Bound (IB&B) is a search algorithm that combines evaluation of parts of the search space to check on their likeliness to contain solutions with domain splitting, which aims to separate solutions. When used for optimization, the objective function is evaluated using interval arithmetic. A sub-box is discarded if the objective function's evaluation shows that it cannot contain a global minimum. After evaluating a box, if it still potentially contains a solution and it is not a small-enough box, it is split into two smaller sub-boxes and these boxes are stored for future review. When a box is not discarded and cannot be split because it is now too small, it is kept aside as a potential solution and not explored again. This exploration-and-sub-division process continues until there is no more box to be explored. The outcome of an IB&B algorithm is a set of narrow boxes that contain all possible values of the parameters for which the objective function reaches it minimal value.

In general, IB&B is improved with contractors. Instead of simply evaluating the objective function and keeping, discarding, or splitting boxes, each explored box undergoes a contraction step that allows to split much smaller boxes or even to discard them altogether. This variation is called Branch & Prune (B&P). Figure 3 shows a general sketch of this algorithm.

---

**Fig. 2.** Domain contraction: $x$ is reduced to $x'$ using constraint $g(x) \leq 0$.

**Fig. 3.** Branch & Prune, algorithm sketch. The sub-domains $x^+$ and $x^-$ are contracted into $x^+_c$ and $x^-_c$ using the constraint-based contractor $g_c$.

---

**III. Speculation in Global Optimization**

In classic IB&B and B&P, search-space partition is an essential component and drive that support reliable results. Both algorithms bisect the interval domain of one variable at a time, traditionally creating two new domains to be searched. There exist different heuristics to select this variable. Common selection methods include round robin, largest-first and smear-based.

In [7], [13], we proposed a different approach, which suggest to focus the splitting on one dimension only: the range of the objective function, and to update it as we discover new values of the objective function. In what follows we recall this approach, propose improvements along with combination strategies.

---

**A. Speculative optimization**

In previous work [7], [13], we presented a speculation-based B&P algorithm for global optimization with a new bisection rule. Our speculative algorithm:

1) First evaluates the objective function using interval arithmetic to obtain a range that contains all possible values for the objective, $f_1 = [L, U]$;
2) Second, instead of bisecting the search space, the algorithm splits \( f_i \) into two sub-ranges, \( [f_i, m(f_i)] \) and \( [m(f_i), \bar{f}] \).

3) Next, the algorithm speculates by “betting” on the lower of the two sub-ranges.
   a) This “bet” becomes a constraint on the objective, which a contractor will enforce to reduce the size of the search space.
   b) The range not selected as a “bet” is saved in case the original bet is wrong, meaning there is no solution in that initial “bet”.

Figure 4 shows the range-splitting part of the algorithm.

In what follows we present our proposed improvement to
the above speculative algorithm along with other strategies to further improve it.

The speculation graft strategy is a hybridization of speculation and the traditional B&P approach. When contraction fails to prune the domain during a speculative process, the algorithm switches to a B&P method. This method is similar to grafting a branch of a tree into the branch stump of another. This B&P graft keeps the “bet” that produced it as initial bounds on the objective, improving or discarding them through domain bisection and contraction. The algorithm returns to speculation only if it discards all the sub-domains created by the B&P graft. Figure 5 provides a general overview of this strategy.

The preconditioning strategy uses an inexpensive local search to set an initial upper bound on the sought optimum value of the objective function. Local search finds a local minimum, which becomes the new upper bound of the initial range of the objective. The algorithm begins speculation bisection using this new smaller range instead of \( \bar{f}(x) \) in the original speculative algorithm.

The Krawczyk contraction strategy uses the interval gradient of the objective function. The gradient of a function \( \nabla f \) is a vector containing the first order partial derivatives of the objective function. The interval gradient results from evaluating interval extensions of the partial derivatives in

2) Second, instead of bisecting the search space, the algorithm splits \( f_i \) into two sub-ranges, \( [f_i, m(f_i)] \) and \( [m(f_i), \bar{f}] \);

3) Next, the algorithm speculates by “betting” on the lower of the two sub-ranges.
   a) This “bet” becomes a constraint on the objective, which a contractor will enforce to reduce the size of the search space.
   b) The range not selected as a “bet” is saved in case the original bet is wrong, meaning there is no solution in that initial “bet”.

Figure 4 shows the range-splitting part of the algorithm.

In what follows we present our proposed improvement to
the above speculative algorithm along with other strategies to further improve it.

The speculation graft strategy is a hybridization of speculation and the traditional B&P approach. When contraction fails to prune the domain during a speculative process, the algorithm switches to a B&P method. This method is similar to grafting a branch of a tree into the branch stump of another. This B&P graft keeps the “bet” that produced it as initial bounds on the objective, improving or discarding them through domain bisection and contraction. The algorithm returns to speculation only if it discards all the sub-domains created by the B&P graft. Figure 5 provides a general overview of this strategy.

The preconditioning strategy uses an inexpensive local search to set an initial upper bound on the sought optimum value of the objective function. Local search finds a local minimum, which becomes the new upper bound of the initial range of the objective. The algorithm begins speculation bisection using this new smaller range instead of \( \bar{f}(x) \) in the original speculative algorithm.

The Krawczyk contraction strategy uses the interval gradient of the objective function. The gradient of a function \( \nabla f \) is a vector containing the first order partial derivatives of the objective function. The interval gradient results from evaluating interval extensions of the partial derivatives in

the gradient. Minimum and maximum points \( x^* \) evaluate to \( \nabla f(x^*) = 0 \). The Krawczyk contraction strategy uses the interval gradient as a system of linear equations and attempts to solve it using the Krawczyk method and its main components, the Krawczyk operator. The Krawczyk operator contracts the domain if there is a unique point solution to the system of equations in that box. Once a domain is contracted with the Krawczyk operator, the algorithm executes the complete Krawczyk method, which returns a box whose width is less than a predefined \( \epsilon \) containing the point solution. If the Krawczyk contractor returns a reduced box, this box becomes a solution candidate. When used in the context of a B&B algorithm, this concludes the exploration of a particular sub-domain, and may set a new bound on the objective. There might be additional sub-domains to explore, as long as these domains can contain an improvement.

IV. Experiments

In this work, we examine the effects of different strategies to improve speculative optimization.

The algorithm is implemented in Python 2.7, using the BigFloat floating-point library to build our custom interval arithmetic library. Functions are evaluated using natural interval arithmetic. The base contractor is HC4. For B&P, a domain is split using Ratz’s bisection [14], which selects a variable \( x_i = [a_i, b_i] \) s.t. \( \|d_i\| (b_i - a_i) \to max \), where \( d_i \) is the interval approximation of the partial derivative of the objective \( f \) on variable \( x_i \).

Local search is done through the SciPy [15] implementation of the BroydenFletcherGoldfarbShanno algorithm for box constraints (L-BFGS-B [16]). The BFGS algorithm is a Quasi-Newton method that uses an approximation of the Hessian matrix of the objective to compute an improvement of the optimum candidate on each Newton iteration, quickly converging to a local solution. The limited-memory box-bound version of the algorithm (L-BFGS-B) uses a smaller set of vectors instead of the entire approximation of the Hessian, and delimits the domain using box constraints. The initial point
selected for the L-BFGS-B algorithm is always the midpoint of the current box.

Six versions of the algorithm are part of this experiment. They are:

- Type A: Traditional IB&P, using interval arithmetic to set the bounds of the objective and HC4 as contractor.
- Type A\(_K\): Same as type A, but with a Krawczyk contraction strategy after the HC4 contraction.
- Type B: Speculation graft strategy, with the first “bet” from an initial interval evaluation of the objective function and using HC4 to contract the domain.
- Type B\(_K\): Same as type B, but with a Krawczyk contraction strategy after the HC4 contraction.
- Type C: Speculation graft strategy combined with the preconditioning strategy and using HC4 to contract the domain.
- Type C\(_K\): Same as type C, but with a Krawczyk contraction strategy after the HC4 contraction.

Each version of the algorithm was tested on a set of 55 unconstrained continuous optimization problems, with \(\epsilon = 10^4\) and a timeout of 1 hour. The results of each execution are compared w.r.t. the quality of the returned interval solution. The solution can be a global minimum (that is, it is an enclosure of the global minimum), an approximation of the global minimum, or a timeout without finding any type of solution. The number of problems for which each of the algorithms found solution are reported in Table I.

<table>
<thead>
<tr>
<th>Type</th>
<th>A</th>
<th>A(_K)</th>
<th>B</th>
<th>B(_K)</th>
<th>C</th>
<th>C(_K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Min</td>
<td>3</td>
<td>19</td>
<td>6</td>
<td>24</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>Approx</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Local Min</td>
<td>27</td>
<td>11</td>
<td>20</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Timeout</td>
<td>25</td>
<td>23</td>
<td>22</td>
<td>19</td>
<td>17</td>
<td>16</td>
</tr>
</tbody>
</table>

Table I: Types of Solutions in Test Results

To measure performance over time, we use the ratio of improvement between different implementations of the algorithm. The ratio of improvement between algorithms \(X\) and \(Y\) is \(R_{XY} = t_X/t_Y\), where \(t_X\) and \(t_Y\) are the execution times for algorithms \(X\) and \(Y\) on a given problem. If \(R_{XY} > 1\), algorithm \(Y\) has better time performance than algorithm \(X\). The closer to 1 the value of \(R_{XY}\) is, the more similar the time performance of both algorithms is. Table II reports comparisons between all versions of the algorithm that implement the Krawczyk contractor on the problems for which those versions found the global optimum.

Finally, Table III shows the ratio of improvement between the preconditioned speculative algorithm with and without Krawczyk method. Ratios in italics represent similar performance for both strategies. Ratios in bold show a considerable improvement of the Krawczyk version over the regular one. In the remaining ratios, the regular speculative version outperforms the Krawczyk version, with the grey background ratios representing outperformance by a great margin (at least 1 to 10).

<table>
<thead>
<tr>
<th>Test Case</th>
<th>N</th>
<th>(A_X) vs (B_K)</th>
<th>(A_K) vs (C_K)</th>
<th>(B_K) vs (C_K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>himmelblau</td>
<td>2</td>
<td>0.9785</td>
<td>3.6157</td>
<td>3.6951</td>
</tr>
<tr>
<td>rotated_1</td>
<td>1</td>
<td>2.0646</td>
<td>0.9509</td>
<td>0.4647</td>
</tr>
<tr>
<td>rotated_16</td>
<td>16</td>
<td>3.0331</td>
<td>2.1261</td>
<td>0.7306</td>
</tr>
<tr>
<td>rotated_32</td>
<td>32</td>
<td>3.0176</td>
<td>2.4689</td>
<td>0.8182</td>
</tr>
<tr>
<td>rotated_64</td>
<td>64</td>
<td>3.0000</td>
<td>2.6414</td>
<td>0.8805</td>
</tr>
<tr>
<td>schwefel_1</td>
<td>1</td>
<td>2.0989</td>
<td>0.9135</td>
<td>0.4352</td>
</tr>
<tr>
<td>schwefel_16</td>
<td>16</td>
<td>2.0900</td>
<td>0.8540</td>
<td>0.4086</td>
</tr>
<tr>
<td>schwefel_2</td>
<td>1</td>
<td>2.1030</td>
<td>0.9017</td>
<td>0.4288</td>
</tr>
<tr>
<td>schwefel_32</td>
<td>32</td>
<td>2.0542</td>
<td>0.8392</td>
<td>0.4085</td>
</tr>
<tr>
<td>schwefel_4</td>
<td>4</td>
<td>2.1497</td>
<td>0.9114</td>
<td>0.4240</td>
</tr>
<tr>
<td>schwefel_5</td>
<td>8</td>
<td>2.0735</td>
<td>0.8781</td>
<td>0.4235</td>
</tr>
<tr>
<td>sphere_1</td>
<td>1</td>
<td>2.0994</td>
<td>0.9823</td>
<td>0.4679</td>
</tr>
<tr>
<td>sphere_16</td>
<td>16</td>
<td>1.9809</td>
<td>0.9777</td>
<td>0.4935</td>
</tr>
<tr>
<td>sphere_2</td>
<td>2</td>
<td>2.1290</td>
<td>0.9946</td>
<td>0.4672</td>
</tr>
<tr>
<td>sphere_32</td>
<td>32</td>
<td>1.9967</td>
<td>0.9980</td>
<td>0.4998</td>
</tr>
<tr>
<td>sphere_4</td>
<td>4</td>
<td>2.0599</td>
<td>0.9883</td>
<td>0.4798</td>
</tr>
<tr>
<td>sphere_64</td>
<td>64</td>
<td>2.0187</td>
<td>1.0035</td>
<td>0.4971</td>
</tr>
<tr>
<td>sphere_8</td>
<td>8</td>
<td>2.0823</td>
<td>0.9754</td>
<td>0.4684</td>
</tr>
</tbody>
</table>

Table II: Comparison of Ratio of Time Improvement Between Algorithms Using Krawczyk Method

Analysis of results

In general, using the Krawczyk method allows to find more global minima or approximations. For the traditional B&P and regular speculation algorithms, Krawczyk provides a considerable improvement on the quality of the solutions, more than doubling the number of problems for which the algorithm returns a global optimum. However, when using a local search preconditioning of the upper bound with speculation (algorithm type C), the improvement is marginal.

In terms of time performance, speculation outperforms the baseline IB&P algorithm when using Krawczyk as a contractor. However, in most cases, the performance of the speculative algorithm with preconditioning is similar to the baseline algorithm. Speculation without preconditioning outperforms the baseline algorithm and the preconditioned speculation 2 to 1.

This curious result led to a performance analysis of preconditioned speculative algorithm with and without Krawczyk. These results are presented in Table III. This shows that in most cases (27 out of 35) preconditioning without Krawczyk outperforms preconditioning with Krawczyk, in some cases (5 out of 10) with a performance ratio of less than 1/10.

V. Conclusion

We presented strategies to improve speculative optimization algorithms, in comparison to optimization algorithms without speculation. We conducted experiments and our results support the fact that the proposed strategies offer improvement of optimization algorithms with and without speculation. In particular, we showed that our speculation graft strategy combined with preconditioning and Krawczyk strategies provides the best results in terms of quality of the solution. In general, using Krawczyk produces more accurate results. However, in many problems that were solved using the Krawczyk strategy, adding the preconditioning strategy yields worse performance than not including it. A closer analysis to the performance of
the algorithms that use speculation graft and preconditioning reveals that in most cases adding the Krawczyk contractor had a negative impact in the time performance.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>N</th>
<th>C vs $C_K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>himmelblau</td>
<td>2</td>
<td>2.713477360</td>
</tr>
<tr>
<td>rastrigin_1</td>
<td>1</td>
<td>0.723588885</td>
</tr>
<tr>
<td>rastrigin_16</td>
<td>16</td>
<td>0.1444402076</td>
</tr>
<tr>
<td>rastrigin_2</td>
<td>1</td>
<td>0.1684221676</td>
</tr>
<tr>
<td>rastrigin_32</td>
<td>32</td>
<td>0.0633155486</td>
</tr>
<tr>
<td>rastrigin_4</td>
<td>4</td>
<td>0.466632008</td>
</tr>
<tr>
<td>rastrigin_64</td>
<td>64</td>
<td>0.023466829</td>
</tr>
<tr>
<td>rastrigin_8</td>
<td>8</td>
<td>0.277736337</td>
</tr>
<tr>
<td>rosenbrock_16</td>
<td>16</td>
<td>30.128826646</td>
</tr>
<tr>
<td>rosenbrock_2</td>
<td>1</td>
<td>4.431719255</td>
</tr>
<tr>
<td>rosenbrock_32</td>
<td>32</td>
<td>6.061914731</td>
</tr>
<tr>
<td>rosenbrock_4</td>
<td>4</td>
<td>16.88563292</td>
</tr>
<tr>
<td>rosenbrock_8</td>
<td>8</td>
<td>7.418682178</td>
</tr>
<tr>
<td>rotated_1</td>
<td>1</td>
<td>0.571058181</td>
</tr>
<tr>
<td>rotated_16</td>
<td>16</td>
<td>0.331458071</td>
</tr>
<tr>
<td>rotated_2</td>
<td>2</td>
<td>0.571951012</td>
</tr>
<tr>
<td>rotated_32</td>
<td>32</td>
<td>0.215770861</td>
</tr>
<tr>
<td>rotated_4</td>
<td>4</td>
<td>0.567991120</td>
</tr>
<tr>
<td>rotated_64</td>
<td>64</td>
<td>0.128064258</td>
</tr>
<tr>
<td>rotated_8</td>
<td>8</td>
<td>0.454410257</td>
</tr>
<tr>
<td>schwefel_1</td>
<td>1</td>
<td>0.549240913</td>
</tr>
<tr>
<td>schwefel_16</td>
<td>16</td>
<td>0.264245282</td>
</tr>
<tr>
<td>schwefel_2</td>
<td>1</td>
<td>0.489488090</td>
</tr>
<tr>
<td>schwefel_32</td>
<td>32</td>
<td>0.243159635</td>
</tr>
<tr>
<td>schwefel_4</td>
<td>4</td>
<td>0.397979901</td>
</tr>
<tr>
<td>schwefel_8</td>
<td>8</td>
<td>0.309797103</td>
</tr>
<tr>
<td>sphere_1</td>
<td>1</td>
<td>0.571939925</td>
</tr>
<tr>
<td>sphere_16</td>
<td>16</td>
<td>0.077392586</td>
</tr>
<tr>
<td>sphere_2</td>
<td>1</td>
<td>0.457359668</td>
</tr>
<tr>
<td>sphere_32</td>
<td>32</td>
<td>0.032752153</td>
</tr>
<tr>
<td>sphere_4</td>
<td>4</td>
<td>0.296781961</td>
</tr>
<tr>
<td>sphere_64</td>
<td>64</td>
<td>0.014708894</td>
</tr>
<tr>
<td>sphere_8</td>
<td>8</td>
<td>0.161188270</td>
</tr>
<tr>
<td>styblinski_tangent</td>
<td>1</td>
<td>1.170557069</td>
</tr>
<tr>
<td>three_hump_camel</td>
<td>2</td>
<td>0.999984375</td>
</tr>
</tbody>
</table>

TABLE III
RATIO OF TIME IMPROVEMENT OF PRECONDITIONED GRAFT SPECULATIVE ALGORITHM WITH AND WITHOUT KRAWCZYK METHOD

We now plan to conduct a detailed study to understand at which specific stages of the algorithm or for which classes of problems Krawczyk should be used for optimal performance. We will also focus on extending this work to constrained optimization.

ACKNOWLEDGMENTS

The work presented here was partially supported by NSF grant CCF No. 0953339.

REFERENCES