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 with no 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 likeliness 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. For rigorous solvers, the challenges lie in reducing the overestimation introduced by interval arithmetic, and improving execution time through new sub-problem generation and selection heuristics.

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 subdivision 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:

$$\min_{x \in \mathbf{x}} f(x) \tag{1}$$

where $f : \mathbf{x} \subset \mathbb{R}^n \to \mathbb{R}$ 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 $\forall x \in \mathbf{x}$, $f(x^*) \leq f(x)$.

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 $x_0 \in [\mathbf{x}]$. 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 $\mathbf{x}$. 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* $\boldsymbol{x_i} = \left[\underline{x_i}, \overline{x_i}\right]$ is the set of all real numbers x such that $\underline{x_i} \leq x \leq \overline{x_i}$. We can $\mathbb{I}$ the set of all such intervals. More generally, we call a *box* $\boldsymbol{X}$ the Cartesian product of intervals: e.g., $\boldsymbol{X} = \boldsymbol{x_1} \times \boldsymbol{x_2} \times \ldots \times \boldsymbol{x_n} \in \boldsymbol{I}^n$.

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

$$\{x \bowtie y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\} \subseteq \boldsymbol{x} \bowtie \boldsymbol{y}$$

Similarly, all real functions f can be "extended" to intervals, as $\boldsymbol{f}$, by systematically extending each of the operators of their symbolic expression to intervals. As a result, we have that:

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

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

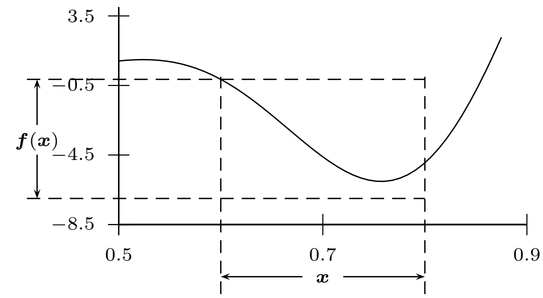

Fig. 1. Interval evaluation of a function

As we can observe on Fig. 1, interval computations allow us to enclose the range of function f over interval $\boldsymbol{x}$, 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 $\boldsymbol{x} = [1, 2]$, then we obtain $\boldsymbol{f}(\boldsymbol{x}) = [-1, 1]$.

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) domains: 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 $\boldsymbol{x}$, we have: $\boldsymbol{x} = C(\boldsymbol{x}) \cap \boldsymbol{x}$.

The desired outcome of this is that either $\boldsymbol{x}$ be reduced or discarded (when $C(\boldsymbol{x}) \cap \boldsymbol{x} = \emptyset$).

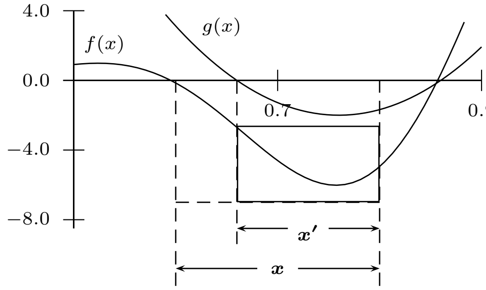

Fig. 2. Domain contraction: $\boldsymbol{x}$ is reduced to $\boldsymbol{x'}$ using constraint $g(x) \leq 0$.

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 $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$. 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 $\boldsymbol{D}$ 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:

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

starting with an initial box $\boldsymbol{D}$.

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:

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

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

$$\mathbf{x}_{k+1} = K(\mathbf{x}_k) \cap \mathbf{x}_k,$$

where $\mathbf{x}_0$ is a starting interval box s.t. $K(\mathbf{x}_0) \subseteq \mathbf{x}_0$, and the solution box is $\mathbf{x}^* = \mathbf{x}_k$ s.t. $\mathbf{x}_{k+1} \approx \mathbf{x}_k$ [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.

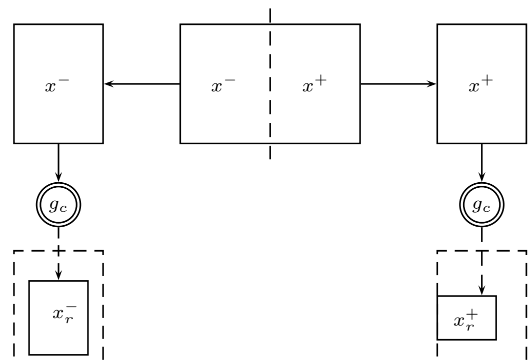

Fig. 3. Branch & Prune, algorithm sketch. The sub-domains x^+ and x^- are contracted into x_r^+ and x_r^- using the constraint-based contractor g_c

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, $f_{\mathbb{I}} = [\underline{f}, \overline{f}]$;

2) Second, instead of bisecting the search space, the algorithm splits $f_{\mathbb{I}}$ into two sub-ranges, $\left[\underline{f}, m\left(f_{\mathbb{I}}\right)\right]$ and $\left[m\left(f_{\mathbb{I}}\right), \overline{f}\right]$;
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.

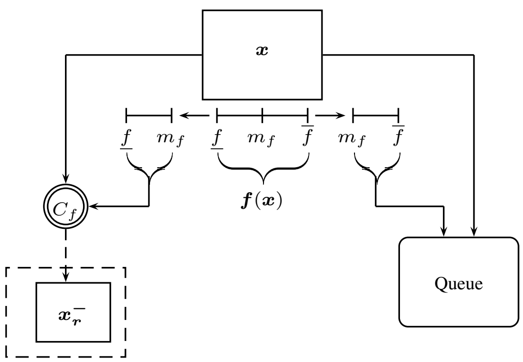

Fig. 4. Speculation algorithm sketch. $\boldsymbol{x}$ is contracted into $\boldsymbol{x}_{\boldsymbol{r}}^{-}$ using the objective function-based contractor C_f with interval $[\underline{f}, m_f]$. $\boldsymbol{x}$ and the interval $[m_f, \overline{f}]$ are stored for possible future exploration.

B. Strategies for speculative optimization

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 $\boldsymbol{f}(\boldsymbol{x})$ in the original speculative algorithm.

The *Krawczyk contraction strategy* uses the interval gradient of the objective function. The *gradient* of a function ∇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

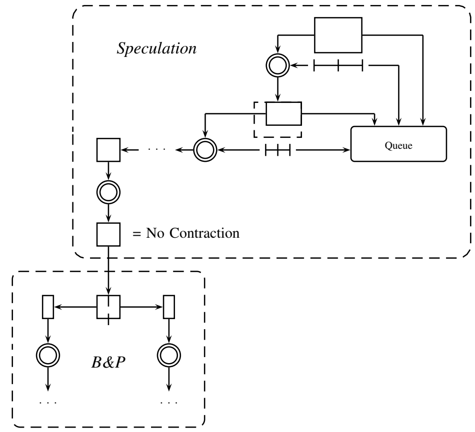

Fig. 5. Overview of the *grafting strategy*. The algorithm stops bisecting the range of the objective if it cannot contract the domain, and starts a B&P sub-tree execution.

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 ϵ 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. $\|d_i\|(b_i - a_i) \rightarrow max$, 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 minimum (an interval whose bounds are less than ϵ away from the global minimum), a local 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.

Type	A	A_K	B	B_K	C	C_K
Global Min	3	19	6	24	36	37
Approx	0	2	7	3	1	2
Local Min	27	11	20	9	1	0
Timeout	25	23	22	19	17	16

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).

Test Case	**N**	**A_K vs B_K**	**A_K vs C_K**	**B_K vs C_K**
himmelblau	2	0.9785	3.6157	3.6951
rotated_1	1	2.0464	0.9509	0.4647
rotated_16	16	3.0331	2.2161	0.7306
rotated_32	32	3.0176	2.4689	0.8182
rotated_64	64	3.0000	2.6414	0.8805
schwefel_1	1	2.0989	0.9135	0.4352
schwefel_16	16	2.0900	0.8540	0.4086
schwefel_2	1	2.1030	0.9017	0.4288
schwefel_32	32	2.0542	0.8392	0.4085
schwefel_4	4	2.1497	0.9114	0.4240
schwefel_8	8	2.0735	0.8781	0.4235
sphere_1	1	2.0994	0.9823	0.4679
sphere_16	16	1.9809	0.9777	0.4935
sphere_2	1	2.1290	0.9946	0.4672
sphere_32	32	1.9967	0.9980	0.4998
sphere_4	4	2.0599	0.9883	0.4798
sphere_64	64	2.0187	1.0035	0.4971
sphere_8	8	2.0823	0.9754	0.4684
AVERAGE		2.1673	1.3395	0.6940

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.

Test Case	**N**	**C vs C$_K$**
himmelblau	2	**2.713477360**
rastrigin_1	1	0.723588885
rastrigin_16	16	0.144440207
rastrigin_2	1	0.614822167
rastrigin_32	32	0.063315548
rastrigin_4	4	0.446632008
rastrigin_64	64	0.023466829
rastrigin_8	8	0.277736337
rosenbrock_16	16	**30.12882664**
rosenbrock_2	1	**4.431719255**
rosenbrock_32	32	**6.061914731**
rosenbrock_4	4	**16.88563292**
rosenbrock_8	8	**7.415862178**
rotated_1	1	0.571058181
rotated_16	16	0.331458071
rotated_2	1	0.671001201
rotated_32	32	0.215177086
rotated_4	4	0.567991102
rotated_64	64	0.128064258
rotated_8	8	0.445410257
schwefel_1	1	0.549240913
schwefel_16	16	0.264245282
schwefel_2	1	0.489488090
schwefel_32	32	0.243159635
schwefel_4	4	0.397079901
schwefel_8	8	0.309797103
sphere_1	1	0.571939925
sphere_16	16	0.07739258
sphere_2	1	0.457359668
sphere_32	32	0.033275215
sphere_4	4	0.296781961
sphere_64	64	0.014708894
sphere_8	8	0.161188270
styblinski_tang_1	1	*1.170557069*
three_hump_camel	2	*0.999984375*

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

[1] M. Tawarmalani and N. V. Sahinidis, "A polyhedral branch-and-cut approach to global optimization," *Math. Program.*, vol. 103, no. 2, pp. 225–249, Jun. 2005.

[2] R. B. Kearfott, "Globsol user guide," *Optimization Methods Software*, vol. 24, no. 4-5, pp. 687–708, Aug. 2009.

[3] J. Merlet, *Alias-c++: An c++ algorithms library of interval analysis for equation systems*, http://bit.ly/ALIAS-C, Accessed: 05-04-2016.

[4] O. Knüppel, "Profil/bias—a fast interval library," *Computing*, vol. 53, no. 3, pp. 277–287, 1994.

[5] G. Trombettoni, I. Araya, B. Neveu, and G. Chabert, "Ibexopt : un module d'optimisation globale sous contraintes fiable," in *13e congrès annuel de la Société française de Recherche Opérationnelle et d'Aide à la Décision*, France, 2012.

[6] A. F. Garcia C., X. Wang, M. Ceberio, R. Bixler, and L. Gutierrez, "Interval optimization to predict software quality assessment decisions," in *Proc. of 2012 INFORMS Optimization Conf.*, Coral Gables, FL, 2012.

[7] A. F. Garcia C., "Contributions to global optimization using interval methods and speculation," M.S. Thesis, The University of Texas at El Paso, Dec. 2014.

[8] S. P. Shary, "Graph subdivision methods in interval global optimization," *Constraint Programming and Decision Making*, pp. 153–170, 2014.

[9] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. Siam, 2009.

[10] M. Ceberio and L. Granvilliers, "Horner's rule for interval evaluation revisited," *Computing*, vol. 69, no. 1, pp. 51–81, Sep. 2002.

[11] I. Araya, B. Neveu, and G. Trombettoni, "An interval extension based on occurrence grouping," *Computing*, vol. 94, no. 2, pp. 173–188, 2012.

[12] F. Benhamou, F. Goualard, L. Granvilliers, and J. Puget, "Revising hull and box consistency," in *Proc. of the 1999 Int. Conf. on Logic Programming*, Cambridge, MA, USA: MIT, 1999, pp. 230–244.

[13] X. Wang and M. Ceberio, "Fuzzy measure extraction for predicting at-risk students," in *Proc. of 2nd World Conference on Soft Computing*, Baku, Azerbaijan, 2012.

[14] V. Kreinovich and R. B. Kearfott, "Where to bisect a box? a theoretical explanation of the experimental results," in *Interval Computations and its Applications to Reasoning Under Uncertainty, Knowledge Representation, and Control Theory. Proc. of MEXICON'98, Workshop on Interval Computations, 4th World Congress on Expert Systems*, 1997.

[15] E. Jones, T. Oliphant, and P. Peterson, *Scipy: Open source scientific tools for python*, http://www.scipy.org, Accessed: 2016-04-30.

[16] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995.