

Dichotomic Search Protocols for Constrained Optimization*

Meinolf Sellmann and Serdar Kadioglu

Brown University, Department of Computer Science
115 Waterman Street, P.O. Box 1910, Providence, RI 02912
`serdark,sello@cs.brown.edu`

Abstract. We devise a theoretical model for dichotomic search algorithms for constrained optimization. We show that, within our model, a certain way of choosing the breaking point minimizes both expected as well as worst case performance in a skewed binary search. Furthermore, we show that our protocol is optimal in the expected and in the worst case. Experimental results illustrate performance gains when our protocols are used within the search strategy by Streeter and Smith.

1 Introduction

In Constrained Optimization, there are two fundamental strategies being used to find and prove optimal feasible solutions. By far the most common strategy is branch-and-bound. By recursively partitioning the problem into sub-problems (“branching”), we systematically cover all parts of the search space. When our objective is to minimize costs, we use a relaxation of the problem to compute an under-estimate of the best solution for a given sub-problem (“bounding”). By comparing this bound with the best previously found solution, we may find that a given sub-problem cannot contain improving solutions, which allows us to discard (or “prune”) the sub-problem from further consideration. There exist a variety of relaxations which can be computed efficiently, the most commonly used is linear relaxation.

Obviously, the efficiency of a branch-and-bound approach depends heavily on the quality of the bounds. For many problems, standard relaxation techniques are reasonably accurate or they can be improved to be reasonably accurate, for example by automatically adding valid inequalities to a linear programming formulation. However, for some problems we have grave difficulty in providing lower bounds that can effectively prune the search. In particular, by exploiting constraint filtering techniques, in Constraint Programming (with few exception such as optimization constraints [5]) the primary focus is on feasibility and not on optimality considerations.

* This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113). We would like to express our thanks to three anonymous reviewers for their helpful comments as well as John Hughes, Anna Lysyanskaya, Claire Mathieu, Steven Smith, and Mathew Streeter for supporting this work and some very insightful discussions.

In order to augment a black-box feasibility solver to handle discrete objective functions, there exists a second strategy known as “dichotomic” or “binary search.” Given an initial interval $[l, u]$ in which the optimal objective value must lie, we can compute the optimum by testing whether a cost lower or equal $l + \lfloor (u - l)/2 \rfloor$ can still be achieved. If so, we continue searching recursively in $[l, l + \lfloor (u - l)/2 \rfloor - 1]$. If not, we know the optimum must lie in $[l + \lfloor (u - l)/2 \rfloor + 1, u]$. When a query to the feasibility solver incurs a cost of T , using classic binary search we can compute the optimum in time $O(T \log(u - l))$.

An implicit assumption in dichotomic search is that positive trials incur the same costs as negative trials. However, based on our empirical knowledge from phase transition experiments [11, 2, 14, 9] we expect that negative trials, where we prove that no better solution exists, are generally more costly than positive trials, where we only need to find one improving solution.

Assume that we are trying to minimize costs within the interval $[0, 100]$, and the true minimum is (seemingly conveniently) 50. A classic binary search hits the optimum immediately, and then attempts to find solutions with objective value lower or equal 24, 37, 43, 46, 48, and 49. While we need to consider the bound 49 in any case to prove optimality of 50, given that a proof of unsatisfiability may be costly, it is unfortunate that binary search considers a rather large number of almost satisfiable instances before 49.

To avoid this situation, we could of course start with an upper bound of 100, and whenever we find a solution with value v only require that from then on we are only interested in solutions with objective value $v - 1$ or lower (see for example the minimization goal in Ilog CP Solver). The downside of this strategy is that we may end up making very slow progress in finding improving solutions.

Our objective is therefore to devise a strategy that allows fast upper bound improvement while avoiding as much as possible costly proofs of unsatisfiability. In particular, we consider *skewed binary searches* [1] where we do not split the remaining objective interval in half but according to a given ratio a . In our example above, assume we use $a = 0.6$ to organize our dichotomic search. Then, we consider 60, 35, 49, 55, 52, 50. Compared to classic binary search, we see that this skewed search considers a number of almost infeasible problems instead of

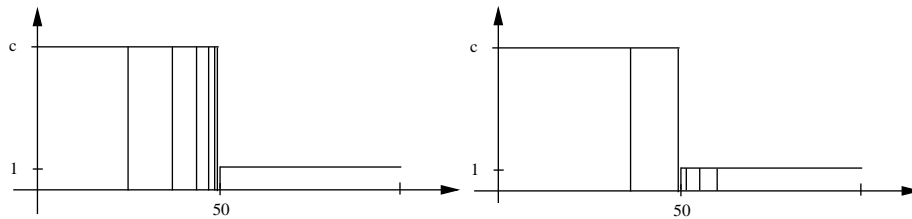


Fig. 1. Dichotomic search for the optimum 50 in the interval $[0, 100]$ when the cost of a negative trial is c and the cost for a positive trial is 1. The left picture illustrates the costs of a classic binary search, the right the costs of a skewed search.

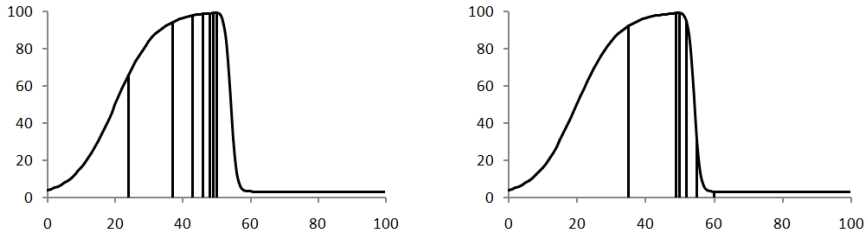


Fig. 2. Dichotomic search for the optimum 50 in the interval $[0,100]$ when the cost of trials follows a typical easy–hard–less-hard pattern. The left picture illustrates the costs of a classic binary search, the right the costs of a skewed search.

almost feasible problems. In Figure 1 we illustrate the costs of classic binary search and the skewed binary search in a model where a negative trial costs of a factor $c \geq 1$ more than a positive trial.

Based on the community’s empirical experience on typical runtime over constrainedness, we expect that finding near-optimal solutions is often significantly easier than proving optimality/infeasibility. In Figure 2 we sketch the two dichotomic searches when assuming a typical curve describing the cost of finding a feasible solution or proving infeasibility for a given upper bound on the objective with the typical easy–hard–less-hard regions (when considering subproblems with increasing constrainedness in the sketch from right to left).

In this paper, we provide dichotomic search protocols for such skewed search problems. In particular, we consider the theoretical model where failures incur costs a factor $c \geq 1$ more than positive trials. For this model, we devise a provably optimal dichotomic search protocol. We then exploit this protocol in a heuristic algorithm which integrates dichotomic search and restarted branch-and-bound. Experimental results on weighted quasi-group and weighted magic square problems illustrate the performance improvements achieved by the new algorithm.

2 Skewed Binary Search

We consider the following theoretical model.

Definition 1. *Given a search-interval $\{l, \dots, u\}$ and a function $f : \{l, \dots, u\} \rightarrow \{0, 1\}$ such that $f(x) = 1 \Rightarrow f(x+1) = 1 \forall l \leq x < u$, we call the problem of finding $y = \min\{x \in \{l, \dots, u\} \mid f(x) = 1\}$ a dichotomic or binary search problem. We call the test whether $f(x) = 1$ for some $x \in \{l, \dots, u\}$ a trial at x . A trial at x is called positive when $f(x) = 1$, otherwise its called negative or a failure. If the cost of a negative trial is c times the cost of a positive trial for some $c \geq 1$, we call c the bias. A binary search problem is called skewed when $c > 1$. An algorithm that makes trials at x to continue its search in $\{x+1, \dots, u\}$ in case of a failure and in $\{l, \dots, x-1\}$ in case of a positive trial is called a (skewed) dichotomic search or a (skewed) binary search. In the case that the search considers trials $x = l + \lfloor a(u-l) \rfloor$ for some constant $a \in [0, 1]$, we call a the balance of the search.*

Theorem 1. *When we assume a uniform distribution of optima in the given interval, the expected effort for a skewed binary search with bias $c \geq 1$ is minimized when setting the balance $a \in [0.5, 1)$ such that*

$$a^c + a = 1.$$

Proof. Let us assume our search interval has length $n \in \mathbb{N}$. According to [19, 15, 1], the expected search cost in a skewed binary tree with balance a is in $\Theta(f(a))$ with¹

$$f(a) := \frac{a + (1-a)c}{-a \log a - (1-a) \log(1-a)} \log(n) + c.$$

Let us denote with $H(a) := -a \log(a) - (1-a) \log(1-a) \in (0, 1]$ the entropy of $a \in (0, 1)$. Then, for the first derivative of f , it holds

$$f'(a) = \left(\frac{(a + (1-a)c)(\log a - \log(1-a))}{H^2(a)} - \frac{(c-1)}{H(a)} \right) \log(n) \quad (1)$$

$$= (a(\log(a) - \log(1-a)) + c \log(a) + cH(a) - cH(a) + H(a)) \frac{\log(n)}{H^2(a)} \quad (2)$$

$$= \frac{(c \log(a) - \log(1-a))}{H^2(a)} \log(n) \quad (3)$$

We note that the sign of the first derivative depends solely on the sign of $c \log(a) - \log(1-a)$. When a satisfies $a^c + a = 1$ then $f'(a) = 0$. For all lower values for $a \in [0.5, 1)$ the derivative is negative, for all larger values it is positive. Consequently, a with $a^c + a = 1$ marks a global minimum of f in the interval $[0.5, 1)$. \square

When our objective is to minimize expected costs under the uniform distribution, the previous theorem tells us how to choose the balance a . The question arises how we should choose a when our goal is to minimize the worst-case performance. Interestingly, we find:

Theorem 2. *The worst-case effort for a skewed binary search with bias $c \geq 1$ is minimized when setting the balance $a \in [0.5, 1)$ such that $a^c + a = 1$.*

Proof. When searching an interval of length $n \in \mathbb{N}$, the worst-case effort of a skewed binary search with balance a is given by the value of the following optimization problem: Maximize $x + cy + c$ such that $a^x(1-a)^y \geq 1/n$, $x, y \geq 0$. We linearize this optimization problem and get

$$\begin{aligned} \max \quad & x + cy + c \\ \text{such that} \quad & \log\left(\frac{1}{a}\right)x + \log\left(\frac{1}{1-a}\right)y \leq \log(n) \\ & x, y \geq 0 \end{aligned}$$

¹ The additional summand c is caused by the fact that we incur costs at nodes and not on branches.

From linear programming theory we know that the maximum is achieved in a corner of this 2-dimensional polytope. The maximum value is thus in

$$\Theta \left(\max \left\{ \frac{1}{\log\left(\frac{1}{a}\right)}, \frac{c}{\log\left(\frac{1}{1-a}\right)} \right\} \log(n) + c \right).$$

Since $-\log(a)$ is strictly monotonically decreasing and $-\log(1-a)$ is strictly monotonically increasing over $[0.5, 1)$, this cost is minimized when choosing the balance $a \in [0.5, 1)$ such that $\frac{1}{\log\left(\frac{1}{a}\right)} = \frac{c}{\log\left(\frac{1}{1-a}\right)}$, which is the same as $\log(1-a) = c \log(a)$, or $1 = a^c + a$. \square

Consequently, we conveniently minimize both expected and worst-case time when setting $a \in [0.5, 1)$ such that $a^c + a = 1$. Then, for the runtime it holds:

Lemma 1. *The expected and worst-case costs of a skewed binary search with bias $c \geq 1$ and balance $a \in [0.5, 1)$ such that $a^c + a = 1$ are in $\Theta \left(c \left(\frac{\log(n)}{\log\left(\frac{1}{1-a}\right)} + 1 \right) \right)$.*

Proof. First, note that $a^c + a = 1$ iff $c = \frac{\log(1-a)}{\log(a)}$. Recall from the proof of Theorem 1 that the expected runtime is in $\Theta(f(a))$ with

$$f(a) = \frac{a + (1-a)c}{-a \log a - (1-a) \log(1-a)} \log(n) + c.$$

Then,

$$f(a) = \frac{a \log(a) + (1-a) \log(1-a)}{(-a \log a - (1-a) \log(1-a)) \log(a)} \log(n) + c \quad (4)$$

$$= \frac{\log(n)}{-\log(a)} + c \quad (5)$$

$$= \frac{c}{\log\left(\frac{1}{1-a}\right)} \log(n) + c \quad (6)$$

Regarding the worst-case runtime, recall from the proof of Theorem 2 that $a^c + a = 1$ implies $\frac{1}{\log\left(\frac{1}{a}\right)} = \frac{c}{\log\left(\frac{1}{1-a}\right)}$. Then,

$$\Theta \left(\max \left\{ \frac{1}{\log\left(\frac{1}{a}\right)}, \frac{c}{\log\left(\frac{1}{1-a}\right)} \right\} \log(n) + c \right) = \Theta \left(\frac{c}{\log\left(\frac{1}{1-a}\right)} \log(n) + c \right).$$

\square

So we essentially gain a factor of $\log\left(\frac{1}{1-a}\right)$ by skewing our search. The question arises how big this factor is in terms of the given bias c .

Lemma 2. *Given $c \geq 1$ and $a \in [0.5, 1)$ such that $a^c + a = 1$, we have that*

$$\log \left(\frac{1}{1-a} \right) \geq \frac{\log(c)}{2}.$$

Proof. Since $a^c + a = 1$ is equivalent with $c = \frac{\log(1-a)}{\log(a)}$, it is sufficient to show that

$$\left(\frac{1}{1-a}\right)^2 \geq \frac{\log(1-a)}{\log(a)},$$

or equivalently that $(1-a)^{(1-a)^2} - a \geq 0$. Let us define $b := 1-a \in (0, 0.5]$, $x := 1/b \geq 2$, and $g(b) := b^{b^2} + b - 1$. Our claim is then equivalent to showing that

$$b^{b^2} + b - 1 = g(b) \geq 0$$

for all $b \in (0, 0.5]$. Consider the first derivative of g :

$$g'(b) = b^{b^2+1}(1 + 2 \ln(b)) + 1.$$

To show that g is monotonically increasing over $(0, 0.5]$, we show that $g'(b) \geq 0$ for all $b \in (0, 0.5]$. Since $1 + 2 \ln(b) < 0$ for all $b \in (0, 0.5]$, it is sufficient to show that $b(1 + 2 \ln(b)) + 1 \geq 0$, or equivalently, that

$$h(x) := 1 + x - 2 \ln(x) \geq 0 \quad \forall x \geq 2.$$

A simple extremum analysis based on the first and second derivative of h shows that h is convex and takes its unique minimum for $x = 2$. Since $h(2) > 0$, we have shown that $g'(b) \geq 0$ over $(0, 0.5]$, and therefore that g is monotonically increasing over $(0, 0.5]$. However, as g approaches 0 from above, we have

$$\lim_{b \rightarrow 0^+} g(x) = \lim_{b \rightarrow 0^+} b^{b^2} + b - 1 \tag{7}$$

$$= \lim_{b \rightarrow 0^+} e^{b^2 \ln(b)} - 1 \tag{8}$$

$$= e^{\lim_{b \rightarrow 0^+} b^2 \ln(b)} - 1 \tag{9}$$

$$= e^0 - 1 = 0. \tag{10}$$

Consequently, $g(b) \geq 0$ for all $b \in (0, 0.5]$. □

With the help of Lemmas 1 and 2, we get immediately:

Theorem 3. *The expected and worst-case costs of a skewed binary search with bias $c \geq 1$ and balance $a \in [0.5, 1)$ such that $a^c + a = 1$ are in $O\left(c \left(\frac{\log(n)}{\log(c)} + 1\right)\right)$.*

To summarize our findings so far: Given a minimization problem where negative trials cost a factor $c \geq 1$ more than positive ones, we minimize the (expected and worst-case) costs of a skewed binary search by choosing the balance $a \in [0.5, 1)$ such that $a^c + a = 1$. With this setting, we essentially gain an asymptotic factor in $\Omega(\log(c))$.

The question arises whether there are other protocols that could minimize the costs further. For example, one may consider a protocol where the balance is not chosen as a constant for the entire search, but that $a \in [0, 1]$ is set in each iteration according to some function over c and also n , the remaining interval length. The following theorem proves that all other dichotomic search protocols cannot perform asymptotically better.

Theorem 4. *Given an interval with length n , considering the breaking point $a \cdot n$ with $a^c + a = 1$ in a skewed binary search with bias $c \geq 1$ is expected optimal in the O -calculus when we assume a uniform distribution of optima in the given interval.*

Proof. Consider a dichotomic search protocol that selects the next trial according to some function $s(c, n)$. For any given interval length $n \in \mathbb{N}$ and bias $c \geq 1$ we show that the expected time that a skewed search using function s takes is greater or equal $\frac{c}{\log(\frac{1}{1-a})} \log(n) + \frac{c}{2}$, where $a^c + a = 1$. We induce over n . For $n = 1$ the claim is trivially true. Now assume $n > 1$ and that the claim holds for all $m < n$. Denote with $p = s(c, n)$ the current trial point. Given that the chance for a positive trial at p is p/n (and $(n-p)/n$ for a negative trial), and by induction hypothesis, for the expected costs it holds that

$$\text{cost}(s, c, n) \geq \frac{n-p}{n} \left(c + \frac{c \log(n-p)}{\log(\frac{1}{1-a})} + \frac{c}{2} \right) + \frac{p}{n} \left(1 + \frac{c \log(p)}{\log(\frac{1}{1-a})} + \frac{c}{2} \right).$$

With $c = \frac{\log(\frac{1}{1-a})}{\log(\frac{1}{a})}$, it follows

$$\text{cost}(s, c, n) \geq c + \frac{\log(n-p)}{\log(\frac{1}{a})} + \frac{p}{n} \left(1 + \frac{\log(p) - \log(n-p)}{\log(\frac{1}{a})} - c \right) + \frac{c}{2} \quad (11)$$

$$= \frac{1}{\log(\frac{1}{a})} \left(\log\left(\frac{n-p}{1-a}\right) + \frac{p}{n} \log\left(\frac{(1-a)p}{a(n-p)}\right) \right) + \frac{c}{2}. \quad (12)$$

Since we wish to show $\text{cost}(s, c, n) \geq \frac{c \log(n)}{\log(\frac{1}{1-a})} + \frac{c}{2} = \frac{\log(n)}{\log(\frac{1}{a})} + \frac{c}{2}$, it is therefore sufficient to show that

$$\log\left(\frac{n-p}{n(1-a)}\right) + \frac{p}{n} \log\left(\frac{(1-a)p}{a(n-p)}\right) \geq 0,$$

or equivalently that

$$\left(\frac{(1-a)p}{a(n-p)}\right)^{\frac{p}{n}} \geq \frac{n(1-a)}{n-p},$$

or that

$$t(a) := \left(\frac{p}{a}\right)^{\frac{p}{n}} - n \left(\frac{1-a}{n-p}\right)^{\frac{n-p}{n}} \geq 0.$$

For the first and second derivation of t we have

$$t'(a) = \left(\frac{1-a}{n-p}\right)^{\left(\frac{n-p}{n}\right)-1} - \frac{1}{n} \left(\frac{p}{a}\right)^{\frac{p}{n}+1} \quad \text{and}$$

$$t''(a) = \frac{p}{a^2 n} \left(\frac{p}{a}\right)^{\frac{p}{n}} \left(\frac{p}{n} + 1\right) + \frac{1}{n-p} \left(1 - \frac{n-p}{n}\right) \left(\frac{1-a}{n-p}\right)^{\frac{n-p}{n}-2}.$$

Clearly, $t''(a) > 0$ for all $a \in [0.5, 1)$, and therefore t is convex on this interval. Furthermore, $t'(\frac{p}{n}) = 0$ and $t(\frac{p}{n}) = 0$, and therefore $t(a) \geq 0$ over $[0.5, 1)$. \square

As a consequence of the previous theorem and Lemma 1, which states that our skewed binary search protocol does not work worse in the worst-case than it does in the expected case, we finally get:

Corollary 1. *Given an interval with length n , considering the breaking point $a \cdot n$ with $a^c + a = 1$ in a skewed binary search with bias $c \geq 1$ is asymptotically optimal in the worst-case.*

3 Skewed Dichotomic Search for Constrained Optimization

The previous theoretical study, while interesting in its own right and applicable in realistic scenarios like the one considered in [1], cannot be exploited directly when considering constrained optimization. This is for various reasons. First of all, as we discussed earlier and illustrated in Figure 2, in optimization practice, failures do not generally incur costs that are a constant factor higher than those of positive trials. Consequently, there is a disconnect between the theoretical model and reality.

The second reason why our protocol is not directly applicable is because, in practice, we do not actually know the factor by which a negative trial is – say, on average – more expensive than a positive trial. We could of course try to estimate such a ratio based on our experience with past trials. However, when the skewed search actually works well we hope to avoid negative trials as best as we can, so the sampling is skewed and there will be very little statistical data to work with. Furthermore, in some cases the lower bounds on the objective that we can compute may be so bad that we may not even strive to find and prove an optimal solution. Instead, our objective may be to compute high quality solutions as quickly as possible.

Finally, in real applications, we may expect that, when a backtracking algorithm finds a new upper bound, there may be other solutions that further improve the objective and can be found quickly when investing only a little more search. Classic branch-and-bound algorithms (to which we will refer as $B+B$), where the current upper bound on the objective is based on the best solution found so far, benefit from such a clustering of good solutions. Note that branch-and-bound can also be parameterized to improve on upper bounds more aggressively. For example, when only an approximately optimal solution is sought, we can set the new upper bound to $(1 - \varepsilon)u$ where $\varepsilon > 0$ and u is the value of the best solution found. Or, following an idea presented in [18], one could set the upper bound for pruning more aggressively based on empirical evidence where the optimal objective may be expected.

3.1 The Streeter-Smith Strategy

To address some of these issues, we follow the work from Streeter and Smith [17] who propose a dichotomic search strategy which considers (potentially incomplete) trials with a given fail-limit. They show that their parameterized strategy

Query strategy $S_3(\beta, \gamma, \rho)$:

1. Initialize $T \leftarrow \frac{1}{\gamma}$, $l \leftarrow 1$, $u \leftarrow U$, $t_l \leftarrow \infty$, and $t_u \leftarrow -\infty$.
2. While $l < u$:
 - (a) If $[l, u - 1] \subseteq [t_l, t_u]$ then set $T \leftarrow \frac{T}{\gamma}$, set $t_l \leftarrow \infty$, and set $t_u \leftarrow -\infty$.
 - (b) Let $u' = u - 1$. If $[l, u']$ and $[t_l, t_u]$ are disjoint (or $t_l = \infty$) then define

$$k = \begin{cases} \lfloor (1 - \beta)l + \beta u' \rfloor & \text{if } (1 - \rho)l > \rho(U - u') \\ \lfloor \beta l + (1 - \beta)u' \rfloor & \text{otherwise;} \end{cases}$$

else define

$$k = \begin{cases} \lfloor (1 - \beta)l + \beta(t_l - 1) \rfloor & \text{if } (1 - \rho)(t_l - l) \\ & > \rho(u' - t_u) \\ \lfloor (1 - \beta)u' + \beta(t_u + 1) \rfloor & \text{otherwise.} \end{cases}$$

- (c) Execute the query $\langle k, T \rangle$. If the result is “yes” set $u \leftarrow k$; if the result is “no” set $l \leftarrow k + 1$; and if the result is “timeout” set $t_l \leftarrow \min\{t_l, k\}$ and set $t_u \leftarrow \max\{t_u, k\}$.

Algorithm 1: The Streeter-Smith Strategy

given in Algorithm 1 achieves an optimal competitive ratio for any fixed set of parameters $0 < \beta \leq 0.5$, $0 < \gamma < 1$, and $0 < \rho < 1$.

Assume we set $\beta = \rho = \gamma = 0.5$. Strategy S_3 then proceeds as follows: It first tries the midpoint of the given interval under some fail-limit. When the trial is inconclusive, the next trial is at $\frac{3}{4}$ of the interval and $\frac{1}{4}$ if the first is also inconclusive. This way, the search points are driven to the borders of the search interval where we expect cheaper trials. If no improved upper and lower bounds are found even for trials at the very border of the interval, the fail-limit is multiplied with $\frac{1}{\gamma}$, and the entire process is repeated. As soon as an improved upper or lower bound is found, the search interval is shrunk accordingly. Note how parameter β shifts the trial point towards the upper bound for lower values of β . Parameter ρ determines the balance how much effort we put on upper-bound rather than lower-bound improvement. In our experiments, negative trials were so costly that the best performance was always achieved by setting $\rho \leftarrow 1$. The parameter γ finally determines how quickly the fail-limit grows. In our experiments, we chose the initial fail-limit as 1000 and $\gamma \leftarrow \frac{2}{3}$. We will refer to this algorithm with the acronym *SS*.

The way how the algorithm proceeds is illustrated in Figure 3. The algorithm sets a fail-limit T and then maintains the current upper and lower bound as well as a time-out interval. The algorithm then performs two interleaved dichotomic searches with bias β , one in the interval $[t, t_l]$, the other in $[t_u, u]$, until the best upper and lower bounds for the given time-limit are achieved. Then, the fail-limit is increased geometrically, and two new dichotomic searchers are initiated.

3.2 Parameter Tuning based on Skewed Binary Search Protocols

While the Streeter-Smith strategy exploits a black-box feasibility solver, the specific solvers that we use for constraint satisfaction are known to benefit from randomization and restarts. Therefore, in a variant of algorithm SS we choose to set the fail-limits in a more continuous fashion than in the Streeter-Smith strategy: After each inconclusive trial, we update the fail-limit linearly to $1000(t+1)$, where t is the number of the last trial that was inconclusive.

With respect to the fact that a backtrack-search may actually yield feasible and potentially improving solutions near a new solution that has been found, we also propose not to stop the search in case of a positive trial. Instead, we choose the next trial point and use this upper bound to prune the search from then on. When we find a new improving solution, we again set the new upper bound aggressively. If we prove unsatisfiability of the new trial or end the search at the initially given fail-limit, we continue in accordance to S_3 .

We observe that the interleaved searches for the best achievable upper and lower bound under some fail-limit depicted in Figure 3 resemble our cost-model from Figure 1. Based on our theoretical study of this cost-model, we are now in a good position to exploit our dichotomic search protocol to tune the parameter β which we propose to choose dynamically for every trial rather than treating it as fixed. Our modified Streeter-Smith strategy works as follows: Whenever we find an improving upper bound, we record how many fails it took within the current restart to produce the new upper bound. Based on these numbers, we keep track of the current average number of failures that it takes to compute a new upper bound. Then, we set the bias c to the ratio of the current fail-limit and this running average, as we expect the search for an improved upper bound to take the running average while a negative trial incurs at most the costs of the current fail-limit.

Of course, for our bias $c \geq 1$, we could approximate $a \in [0.5, 1)$ online. The algorithm will be faster, however, when we pre-compute the corresponding a -values for realistic values of c , say for all natural numbers lower than 1000. In our implementation, we pre-computed values for a corresponding to c which grows exponentially starting at 1 by setting $c_{t+1} := c_t(1 + \varepsilon)$ for some small $\varepsilon > 0$. For a concrete c , we then interpolate the value for a . The parameter β is then dynamically set to $\beta \leftarrow 1 - a$.

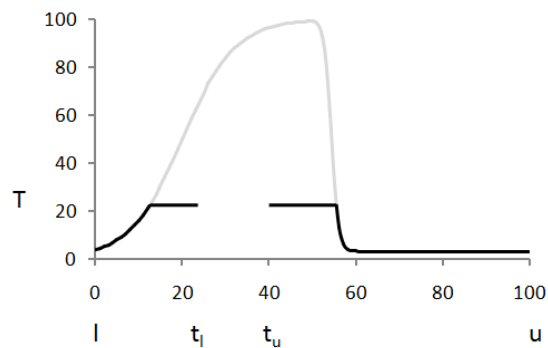


Fig. 3. The Streeter-Smith strategy for constrained optimization on the interval $[1,100]$.

Query strategy $SS-lc-skewed(l, u, \delta)$

1. Initialize $f \leftarrow 0, s \leftarrow 0, avg \leftarrow 1, T \leftarrow \delta, T' \leftarrow T, timeout \leftarrow false, l' \leftarrow l$.
2. While $l < u$:
 - (a) Let $u' \leftarrow u - 1$
 - (b) If $timeout = true$ and $l' \geq u'$, then set $timeout \leftarrow false$ and $l' \leftarrow l, T \leftarrow T + \delta, T' \leftarrow T$
 - (c) Let $\beta \leftarrow 1 - a[T/avg]$
If $timeout = true$, then set $k \leftarrow l' + (u' - l') * \beta$ else set $k \leftarrow l + (u' - l) * \beta$
 - (d) Execute a limited randomized backtrack search with parameters $\langle in : k, in : T', out : failures, out : bestSol \rangle$.
 - (e)
 - i. If the result is “yes”
Set $u \leftarrow bestSol, s \leftarrow s + 1, f \leftarrow f + failures, T' \leftarrow T - failures, avg \leftarrow f/s, \beta \leftarrow 1 - a[T/avg], k \leftarrow l + (u - l) * \beta, timeout \leftarrow false$ and $l' \leftarrow l$. Continue the latest backtrack search with parameters $\langle in : k, in : T', out : failures, out : bestSol \rangle$. Go back to (e)
 - ii. If the result is “no”
Set $l \leftarrow k + 1, T \leftarrow T + \delta, T' \leftarrow T, timeout \leftarrow false$ and $l' \leftarrow l$.
 - iii. If the result is “timeout”
Set $l' \leftarrow k, T \leftarrow T + \delta$ and $T' \leftarrow T, timeout \leftarrow true$.

Algorithm 2: Skewed Restarted Search

Depending on whether we use a specific β or our skewed protocol $\beta = 1 - a$ we refer to this variation of the Streeter-Smith strategy with the acronym $SS-lc$ or $SS-lc-skewed$, respectively. The latter is outlined in Algorithm 2: Given a search interval $[l, u]$, as well as an increment unit δ to update the successive fail-limits T , the average number of failures to compute a new upper bound, avg , is initialized to 1 and the trial point k , is determined by the skewing parameter, $\beta \leftarrow 1 - a[T/avg]$, where $a[T/avg]$ gives the skewing parameter a for bias T/avg . The algorithm performs a search with fail-limit T' , and returns the number of failures along with a new upper bound, $bestSol$, if a solution is ever found. If the search for an improving solution is successful, we decrease the upper bound u , increase the number of successful trials s as well as the total number of failures f , and reset the timeout flag. Then, the backtrack search is continued with the updated values of β, k and T' . If the search proves that no solution with costs lower or equal k exists, we increase the lower bound l , and the fail-limit and reset the time-out flag. If the query result is “timeout”, the timeout flag is set to $true$, a temporary lower bound l' is set to k , and the fail-limit is increased. During the search, if the temporary lower bound meets the upper bound, we reset the timeout flag and restart the search from the lower bound l with a linearly increased fail-limit T . This entire process is repeated until the search interval is consumed. To facilitate the presentation, we only show the modified upper bound improvement here. Just as in the Streeter-Smith strategy, we can of course interleave the while-loop in step (2) with another skewed search that aims at increasing the lower bound quickly.

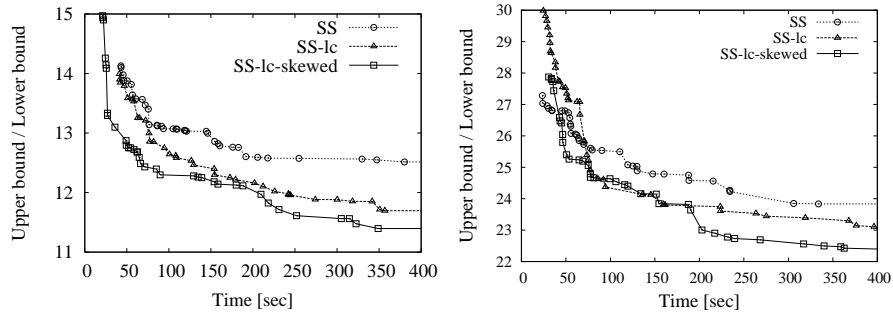


Fig. 4. Comparison of SS, SS-lc, and SS-lc-skewed on weighted magic square problems. We show the average ratio of upper to lower bound for 20 instances with 36 (left) and 64 cells (right) and objective weights p_{ij} for each cell (i, j) chosen uniformly in $[1, 36]$ and $[1, 64]$, respectively.

4 Numerical Results

In [1], skewed dichotomic search has been thoroughly investigated in the context of sorting. Here, branch-prediction and cache-misses can cause a skewed search to work more efficiently than classic binary search. Experimental results show that skewing the search leads to gains in the order of around 15%. To assess the effect of skewing dichotomic search for constrained optimization, in this section we compare the three algorithms outlined above on two benchmark problems, the weighted quasigroup-completion problem and the weighted magic square problem.

Definition 2. [Weighted Quasigroup Completion] *Given a natural number $n \in \mathbb{N}$, a quasigroup Q on symbols $1, \dots, n$ is an $n \times n$ matrix in which each of the numbers from 1 to n occurs exactly once in each row and in each column. We denote each element of Q by q_{ij} , $i, j \in \{1, 2, \dots, n\}$. n is called the order of the quasigroup. Given profit values $p_{ij} \in \mathbb{N}$, $i, j \in \{1, 2, \dots, n\}$, and a set of tuples $F = \{(k, i, j) \mid 1 \leq i, j, k \leq n\}$, the Weighted Quasigroup Completion problem consists in computing a quasigroup Q such that $q_{ij} = k$ for all $(k, i, j) \in F$ and the value $\min_i \sum_j p_{ij} q_{ij}$ is minimized.*

Definition 3. [Weighted Magic Square Problem] *Given a natural number $n \in \mathbb{N}$, a magic square M of order n is an $n \times n$ matrix in which each of the numbers from 1 to n^2 occurs exactly once and such that the sum of all values in each row, column, and main diagonal are identical. We denote each element of M by m_{ij} , $i, j \in \{1, 2, \dots, n\}$. Given profit values $p_{ij} \in \mathbb{N}$, $i, j \in \{1, 2, \dots, n\}$, the Weighted Magic Square Problem consists in computing a magic square M such that the value $\min_i \sum_j p_{ij} m_{ij}$ is minimized.*

From the perspective of the Constraint Programming (CP), Artificial Intelligence (AI), and Operations Research (OR) communities, combinatorial design problems as the ones given above are interesting as they are easy to state but possess rich structural properties that are also observed in real-world applications such as scheduling, timetabling, and error correcting codes. Thus, the area of combinatorial designs has been a good source of challenge problems for these research communities. In fact, the study of combinatorial design problem instances has pushed the development of new search methods both in terms of systematic and stochastic procedures. For example, the question of the existence and non-existence of certain quasigroups with intricate mathematical properties gives rise to some of the most challenging search problems in the context of automated theorem proving [20]. So-called general purpose model generation programs, used to prove theorems in finite domains, or to produce counterexamples to false conjectures, have been used to solve numerous previously open problems about the existence of quasigroups with specific mathematical properties. Considerable progress has also been made in the understanding of symmetry breaking procedures using benchmark problems based on combinatorial designs [3, 6, 8, 16]. The study of search procedures on benchmarks based on quasigroups has led to the discovery of the non-standard probability distributions that characterize complete (randomized) backtrack search methods, so-called heavy-tailed distributions [7].

For the purpose of testing dichotomic search protocols, the chosen benchmarks are interesting since even finding feasible solutions only is already hard. Moreover, it is a challenge to provide tight bounds on the objective, which is exactly when experts usually revert to a dichotomic search to solve a problem.

Our results are illustrated in Figures 4 and 5. Experiments were run on an AMD Athlon 64 X2 Dual Core Processor 3800+ using Ilog Solver 6.5. For both problems, the CP-models used to solve particular queries are based on the obvious AllDifferent constraints. We fill the squares row by row, whereby the row to be filled next is determined by the row that currently marks the lower bound on the objective. Within a row, we pick a random variable with minimal domain and assign the lowest value in its domain first. All dichotomic algorithms perform an initial improvement phase where we try to quickly tighten the initial search interval as best as possible. Because of the difficulty to find even feasible solutions only, we did not use local search for this purpose, but a number of short, restarted tree-searches with a tight fail-limit.

The pure B+B approach without restarts often fails to provide feasible solutions within the given time-frame. Consequently, we do not show the results for this method in the figures. We believe that the inferior performance of B+B is due to the fact that it conducts one continuous search that is not restarted. Thus, it gets easily stuck in an area of the search space which does not contain feasible and improving solutions. This trap is particularly big as the CP domain-based lower bounds available to our algorithms are not of very high quality. All other algorithms avoid this problem by exploiting the benefits of a somewhat randomized branching variable selection with frequent restarts.

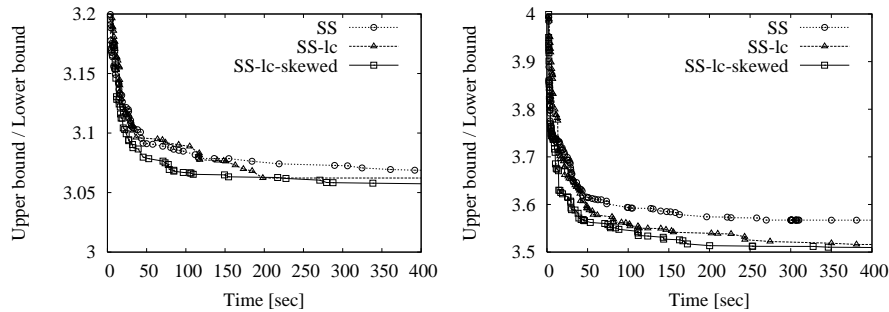


Fig. 5. Comparison of SS, SS-lc, and SS-lc-skewed on weighted quasigroup-completion problems. We show the average ratio of upper to lower bound for 20 instances with 100 (left) and 144 cells (right) and objective weights p_{ij} for each cell (i, j) chosen uniformly in $[1, 100]$ and $[1, 144]$, respectively.

With respect to the remaining algorithms, we observe that SS-lc works better than the pure Streeter-Smith strategy SS. That is, we find that continuously updating the fail-limit and continuing the search with an improved upper bound after a new solution has been found is beneficial for constrained optimization. In the B+B approach, when it does find a solution, we often find that more improving solutions are found shortly afterwards. We believe that this clustering of solutions in some small subtree is caused by the algorithm having found a desirable partial assignments. Such a clustering is exploited by SS-lc by continuing the search rather than restarting directly.

Finally, we see that SS-lc-skewed leads to an additional improvement. In this method, the fact that the Streeter-Smith strategy considers strict fail-limits allows us to get a good estimate on the search-bias c . As we had hoped, using an optimistic but not overly aggressive way to set new upper bounds based on this estimate of the bias and our theoretically optimal setting allows us to find improving solutions faster and thereby close the gap between upper and lower bound more rapidly.

5 Conclusions

We studied a theoretical model for dichotomic search algorithms and devised a protocol which minimizes both expected as well as worst case performance in a skewed binary search. Furthermore, we showed that our protocol is optimal in the expected and in the worst case. Earlier experiments in the sorting domain by Brodal and Moruz had already shown practical gains from skewing binary search algorithms. In the context of constrained optimization, by exploiting the strategy proposed by Streeter and Smith, dichotomic search can be exploited in practice while skewing the search leads to faster improvements of the upper bound in constrained minimization.

References

1. G.S. Brodal and G. Moruz. Skewed Binary Search Trees. *ESA*, pp.708–719, 2006.
2. J.A. Crawford and L.D. Auton. Experimental Results on the Cross-Over Point in Random 3-SAT. *Artificial Intelligence*, 81:31–57, 1996.
3. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry Breaking. *7th International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 2239:93–107, 2001.
4. S. Ferr and R.D. King. A dichotomic search algorithm for mining and learning in domain-specific logics. *Fundamenta Informaticae*, 66(1–2):1–32, 2005.
5. F. Focacci, A. Lodi, and M. Milano. Cost-Based Domain Filtering. *5th International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 1713:189–203, 1999.
6. F. Focacci and M. Milano. Global Cut Framework for Removing Symmetries. *CP*, pp. 77–92, 2001.
7. C.P. Gomes, B. Selman, N. Crato, H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Automated Reasoning*, 24(1–2):67–100,2000.
8. W. Harvey. Symmetry Breaking and the Social Golfer Problem. *SymCon*, 2001.
9. T. Hogg, B.A. Huberman, and C.P. Williams. Phase Transitions and Complexity. *Artificial Intelligence*, 81, 1996.
10. N. Jussien and O. Lhomme. Dynamic Domain Splitting for Numeric CSPs. *ECAI*, pp.224–228, 1998.
11. S. Kirkpatrick and B. Selman. Critical Behavior in the Satisfiability of Random Boolean Expressions. *Science*, 264:1297–1301, 1994.
12. G.T. Leong. Constraint Programming for the Traveling Tournament Problem. *Project Thesis*, National University of Singapore, 2003.
13. I. Lustig and J.-F. Puget. Constraint Programming. *Encyclopedia of Operations Research and Management Science*, pp.136–140, Kluwer, 2001.
14. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature* 400:133–137, 1999.
15. J. Nievergelt and E.M. Reingold. Binary search trees of bounded balance. *Annual ACM Symposium on Theory of Computing*, pp. 137–142, 1972.
16. B. Smith. Reducing Symmetry in a Combinatorial Design Problem. *CPAIOR*, pp. 351–360, 2001.
17. M. Streeter and S.F. Smith. Using Decision Procedures Efficiently for Optimization. *ICAPS*, pp.312–319, 2007.
18. D. Wojtaszek and J.W. Chinneck. Faster MIP Solutions via New Node Selection Rules. *INFORMS*, 2007.
19. C.K. Wong and J. Nievergelt. Upper Bounds for the Total Path Length of Binary Trees. *Journal of the ACM*, 20(1):1–6,1973.
20. H. Zhang. Specifying Latin Square Problems in Propositional Logic. *Automated Reasoning and Its Applications*, MIT Press, 1997.