Lattice cryptography: from cryptanalysis to New Foundations

Woerden, W.P.J. van

CHAPTER 5

The Closest Vector Problem with Preprocessing

This chapter is based on the joint work 'The randomized slicer for CVPP: sharper, faster, smaller, batchier', with Léo Ducas and Thijs Laarhoven, published at PKC 2020.

5.1 Introduction

The Closest Vector Problem (CVP) can be viewed as an inhomogeneous version of the Shortest Vector Problem (SVP). It asks to compute a closest lattice vector to a given target vector in the real span of the lattice. This is also sometimes referred to as *decoding* the target. These two problems are closely related, and just as for SVP, solving CVP efficiently would directly break most lattice-based cryptographic schemes; e.g., decoding an encrypted message, or forging a signature often essentially boils down to recovering a close(st) lattice point. In many cases it is sufficient to solve an approximate version γ-CVP, where one has to compute a lattice vector that lies at most a factor γ further away than a closest one.

CVP is no easier than SVP, there exists an efficient reduction from approx-SVP to approx-CVP that preserves both the lattice dimension and approximation factor [GMSS99]. In fact most lattice problems have such a dimension-preserving reduction to CVP [Mic08; BN09; MV13]. Provable worst-case reductions in the other direction are less straightforward and tight, and often increase both the dimension and approximation factors.

Still from an average-case perspective, most algorithms for SVP can be adapted to work for CVP with about the same (heuristic) complexity. I.e., the current fastest approaches for solving CVP are just as SVP based on lattice sieving [AKS01; BDGL16; Alb+19; DSW21] and lattice enumeration [Kan83; FP85; GNR10; AN17; ANS18], where the former offers a better asymptotic scaling of the time complexity in terms of the lattice dimension, at the cost of an exponentially large memory consumption.

In contrast to SVP however, CVP receives as input, next to a lattice (basis), a target vector. Even though solving a single instance is as hard as SVP, one could wonder if we can amortize the cost by decoding many targets over the same lattice.

The closest vector problem with preprocessing (CVPP). The closest vector problem with preprocessing (CVPP) is a variant of CVP, where the solver is allowed to perform some preprocessing on the lattice at no additional cost, before being given the target vector. Closely related to this is batch-CVP, where many CVP instances on the same lattice are to be solved; if an efficient global preprocessing procedure can be performed using only the lattice as input, and that would help reduce the costs of single CVP instances, then this preprocessing cost can be amortized over many problem instances to obtain a faster algorithm for batch-CVP. This problem of batch-CVP most notably appears in the context of lattice enumeration for solving SVP or CVP, as a fast batch-CVP algorithm would potentially imply faster SVP and CVP algorithms based on a hybrid of enumeration and such a CVPP oracle [GNR10; DLW20a].

Voronoi cells and the iterative slicer. One method for solving CVPP is the iterative slicer by Sommer–Feder–Shalvi [SFS09]. Preprocessing consists of computing a large list of lattice vectors, and a query is

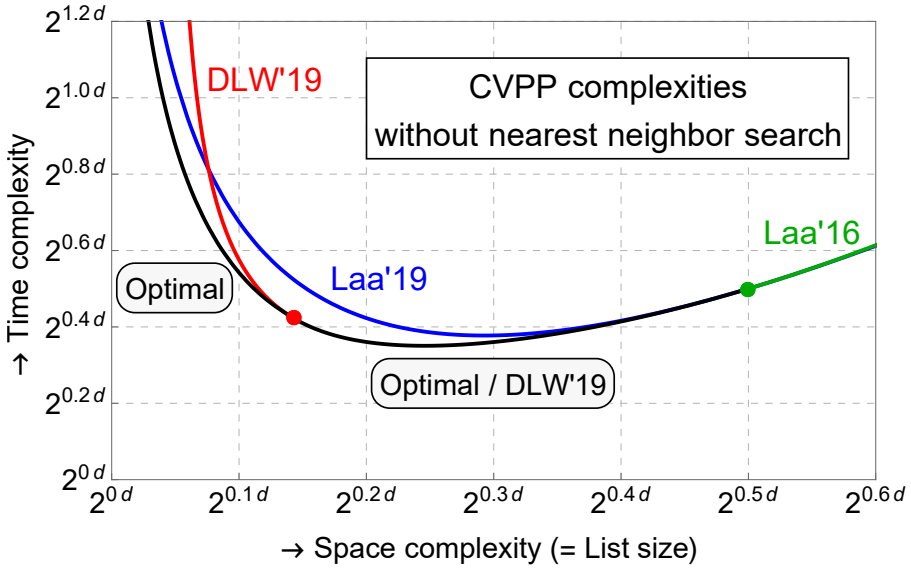


Figure 5.1: Query complexities for solving CVPP without nearest neighbour techniques. The blue curve refers to [Laa19], the red curve to [DLW19], the green curve to [Laa16], and the black curve is the result of our refined analysis. The red point indicates the point where red and black curves merge into one.

processed by "reducing" the target vector $\mathbf{t}$ with this list, i.e. repeatedly translating the target by some lattice vector until the shortest representative $\mathbf{t}'$ in the coset of the target vector is found. The closest lattice vector to $\mathbf{t}$ is then given by $\mathbf{t} - \mathbf{t}'$, which lies at distance $\|\mathbf{t}'\|$ from $\mathbf{t}$. For this method to provably succeed, the preprocessed list needs to contain all $O(2^d)$ so-called Voronoi relevant vectors of the lattice, which together define the boundaries of the Voronoi cell of the lattice. This leads to a $4^{d+o(d)}$ algorithm by bounding the number of reduction steps by $2^{d+o(d)}$ [MV13], which was later improved to an expected time of $2^{d+o(d)}$ by randomizing the algorithm such that the number of expected steps is polynomially bounded [DB15].

Approximate Voronoi cells and the randomized slicer. The large number of Voronoi relevant vectors of a lattice, needed for the iterative slicer to be provably successful, makes the straightforward application of this method impractical and does not result in an improvement over

the best (heuristic) CVP complexities without preprocessing. Therefore we fall back on heuristics to analyse the iterative slicer, as they often better represent the practical complexities of an algorithm than the proven worst-case bounds. Laarhoven [Laa16] proposed to use a smaller preprocessed list of size $2^{d/2+o(d)}$ containing all lattice vectors up to some radius, while heuristically retaining a constant success probability of finding the closest vector with the iterative slicer. Doulgerakis–Laarhoven–De Weger [DLW19] formalized this method in terms of approximate Voronoi cells, and proposed an improvement based on rerandomizations. Rather than hoping to find the solution in one run of the iterative slicer, which would require a preprocessed list of size at least $2^{d/2+o(d)}$, the algorithm uses a smaller list and runs the same reduction procedure many times. Each time starting with a randomly sampled members from the coset of the target vector. The success probability of this randomized slicing procedure, which depends on the size of the list, determines how often it has to be restarted, and thus plays an important role in the eventual time complexity of the algorithm. Doulgerakis–Laarhoven–De Weger [DLW19] obtained a heuristic lower bound on the success probability of this randomized slicer, which Laarhoven [Laa19] later improved upon in the low-memory regime. In particular, these two bounds cross each-other, which suggests that there should be a better (sharp) global bound. Thus the question remains open, what is the actual asymptotic success probability of the randomized slicing procedure, and therefore what is the actual asymptotic time complexity of the current state-of-the-art heuristic method for solving CVPP.

5.1.1 Contributions

Success probability asymptotics via random walks

The main contribution of this work is to answer the central open problem resulting from the approximate Voronoi cells line of work – giving sharp (heuristic) asymptotics on the success probability of the randomized slicer when using a list of the $\alpha^{d+o(d)}$ shortest lattice vectors. To find these sharp bounds, in Section 5.3 we show how to model the flow of the algorithm as a random walk on the coset of the lattice corresponding to the target vector, and we heuristically characterise transition probabilities between different states in this infinite graph.

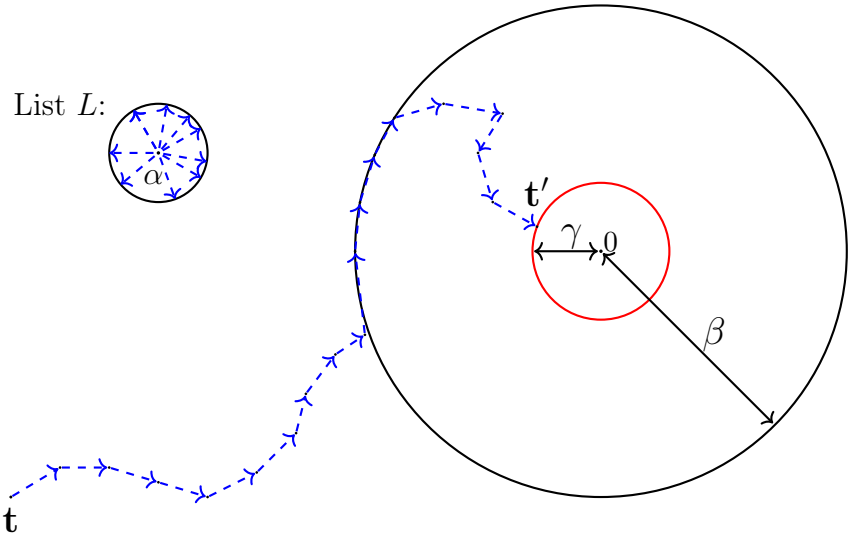


Figure 5.2: The iterative slicer as a random walk over the coset $\mathbf{t} + \mathcal{L}$ using the list of lattice vectors $L = \mathcal{L} \cap \mathcal{B}(\vec{0}, \alpha)$.

We show that the transition probabilities only depend on the norm of coset representatives and therefore the graph can be simplified to an interval $(0, \infty)$ representing the norm. The aforementioned problem of finding the success probability of the slicer then translates to: what is the probability in this graph of starting from a given initial norm and ending at any target norm of at most γ? From [DLW19] we know that we almost always reach a state of norm at most some $\beta = f(\alpha) \geq \gamma$ – reaching this state occurs with probability at least $1/\operatorname{poly}(d)$. However, reaching a state $\beta' < \beta$ occurs only with exponentially small probability $2^{-\Theta(d)}$. Now, whereas the analysis of [DLW19] can be interpreted as lower-bounding the success probability by attempting to reach the target norm in a single step after reaching radius β, we are interested in the arbitrary-step transition probabilities from β to at most γ, so as to obtain sharp bounds.

As every path in our graph from β to γ has an exponentially small probability in d, the total success probability is dominated by that of the highest probable path for large d; which after an appropriate log-transform boils down to a shortest path in a graph. Therefore obtaining the success probability of the randomized slicer is reduced

to determining a shortest path in this infinite graph. We show in Section 5.4 how we can approximately compute this shortest path numerically, using a suitably dense discretization of the search space or using convex optimization. In Section 5.5 we go a step further by proving an exact analytic expression of the shortest path, which results in sharp asymptotics on the success probability of the randomized slicer for the general case of approx-CVP.

Heuristic claim 74 (Success probability of the randomized slicer). *Given a list L of the $\alpha^{d+o(d)}$ shortest lattice vectors as input, the success probability p_{α^2,γ^2} of one iteration of the randomized slicer for γ-CVPP equals:*

$$p_{\alpha^2,\gamma^2} = \prod_{i=1}^{n} \left(\alpha^2 - \frac{(\alpha^2 + x_{i-1} - x_i)^2}{2x_{i-1}} \right)^{d/2+o(d)}$$

with n defined by equation (5.1) on page 154, and x_i as in Definition 81 depending only on α and γ.

Running the randomized slicer for $O(p_{\alpha^2,\gamma^2}^{-1})$ iterations, we expect to solve γ-CVPP with constant probability. Together with a (naive) linear search over the preprocessed list, this directly leads to explicit time and space complexities for a plain version of the randomized slicer for solving CVPP, described in Figure 5.1. When using a large list of size at least $2^{0.1436d+o(d)}$ from the preprocessing phase of CVPP, we derive that one step is optimal, thus obtaining the same asymptotic complexity as [DLW19]. When using less than $2^{0.1436d+o(d)}$ memory we gradually see an increase in the optimal number of steps in the shortest path, resulting in ever-increasing improvements in the resulting asymptotic complexities for CVPP as compared to [DLW19].

Extensions in full work

In the full version[1] of this work we exhibit the versatility of the random walk model. For example we show how to adapt the graph slightly to analyse the success probability of the iterative slicer for the BDD-variant of CVP, where the target lies unusually close to the lattice. In addition, by using a similar random walk model, we derive a tight

[1] Full version: https://eprint.iacr.org/2020/120.pdf

probability analysis of graph-based Nearest Neighbour Search techniques, and the resulting time-memory trade-off when applied to sieving algorithms, improving on previous upper bounds.

Next to the versatility we also further improve the time-memory trade-off by using the asymptotic best bucketing technique [BDGL16] as explained in Section 3.3. While in previous works [Laa16; Laa19] these bucketing techniques came at the cost of large memory overheads, we show how this can be achieved efficiently with only a small memory overhead for CVPP, and no overhead for batch-CVPP.

5.1.2 Working heuristics

The first heuristic which we use is the commonly used Gaussian heuristic, which predicts the number of lattice vectors and their density within certain regions based on the lattice volume. Its use for analysing sieve-type algorithms is well established [NV08; BGJ15; BDGL16; Laa16] and seems consistent with various experiments conducted in the past.

The second heuristic assumption we use is also central in previous work on the randomized iterative slicer [DLW19; Laa19], and consists of assuming that the input target can be randomized, yielding essentially independent experiments each time we randomize the input over the coset of the target vector. Practical experiments from [DLW19] seem to support this assumption.

5.2 The randomized iterative slicer

The iterative slicer (Algorithm 4) is a simple but effective algorithm that aims to solve the closest vector problem or variants thereof. It starts with a preprocessing stage that only depends on the input lattice, after which it can solve multiple problem instances over this fixed lattice, amortizing the cost of preprocessing.

5.2.1 The algorithm

The preprocessing consists of finding and storing a list $L \subset \mathcal{L}$ of lattice vectors. Then given a target point $\mathbf{t} \in \mathbb{R}^d$ the iterative slicer tries to reduce the target $\mathbf{t}$ by the list L to some smaller representative

Algorithm 4: The iterative slicer of [SFS09]

Input: A target vector $\mathbf{t} \in \mathbb{R}^d$, a list $L \subset \mathcal{L}$.
Output: A close vector $\mathbf{v} \in \mathcal{L}$ to $\mathbf{t}$.

Function `IterativeSlicer(`L, $\mathbf{t}$`):`
  $\mathbf{t}_0 \leftarrow \mathbf{t}$;
  for $i \leftarrow 0, 1, 2, \ldots$ **do**
    $\mathbf{t}_{i+1} \leftarrow \min_{\mathbf{v} \in L \cup \{\mathbf{0}\}} \{\mathbf{t}_i - \mathbf{v}\}$;
    if $\mathbf{t}_{i+1} = \mathbf{t}_i$ **then return** $\mathbf{t}_0 - \mathbf{t}_i$;
  end

$\mathbf{t}' \in \mathbf{t} + \mathcal{L}$ in the same coset of the lattice. This is repeated until the reduction fails or until the algorithm succeeds, i.e., when $\|\mathbf{t}'\| \leq \gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$. We then obtain the lattice point $\mathbf{t} - \mathbf{t}'$ that lies at distance at most $\gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$ to $\mathbf{t}$. Observe that $\mathbf{t}'$ is a shortest vector in $\mathbf{t} + \mathcal{L}$ if and only if $\mathbf{v} = \mathbf{t} - \mathbf{t}' \in \mathcal{L}$ is a closest lattice vector to $\mathbf{t}$.

To provably guarantee that the closest vector is found we need the preprocessed list L to contain all the Voronoi-relevant vectors; the vectors that define the Voronoi cell of the lattice. However most lattices have $O(2^d)$ relevant vectors, which is too much to be practically viable. Under the Gaussian heuristic, Laarhoven [Laa16] showed that $2^{d/2+o(d)}$ short vectors commonly suffice for the iterative slicer to succeed with high probability, but this number of vectors is still too large for any practical algorithm. The randomized slicer (Algorithm 5) of Doulgerakis–Laarhoven–De Weger [DLW19] attempts to overcome this large list requirement by using a smaller preprocessed list together with rerandomizations to obtain a reasonable probability of finding a close vector – the success probability of one run of the iterative slicer might be small, but repeating the algorithm many times using randomized inputs from $\mathbf{t} + \mathcal{L}$, the algorithm then succeeds with high probability, without requiring a larger preprocessed list.

Because we can only use a list of limited size, one can ask the question which lattice vectors to include in this list L. Later in the analysis it will become clear that short vectors are more useful to reduce a random target, so it is natural to let L consist of all short vectors up to some radius. Let $\alpha > \lambda_1(\mathcal{L})$ be this radius and denote

Algorithm 5: The randomized iterative slicer of [DLW19]

Input: A target vector $\mathbf{t} \in \mathbb{R}^d$, a list $L \subset \mathcal{L}$, a target distance $\gamma \in \mathbb{R}$.

Output: A close vector $\mathbf{v} \in \mathcal{L}$, s.t. $\|\mathbf{t} - \mathbf{v}\| \leq \gamma$.

Function `RandomizedSlicer`(L, $\mathbf{t}$, γ)**:**
  repeat
    $\mathbf{t}' \leftarrow$ `Sample`($\mathbf{t} + \mathcal{L}$);
    $\mathbf{v} \leftarrow$ `IterativeSlicer`(L, $\mathbf{t}'$);
  until $\|\mathbf{t}' - \mathbf{v}\| \leq \gamma$;
  return $\mathbf{v} + (\mathbf{t} - \mathbf{t}')$;

its square by $a := \alpha^2$. The preprocessed list then becomes

$$L_a := \{\mathbf{x} \in \mathcal{L} : \|\mathbf{x}\|^2 \leq a\}.$$

Throughout this work let us assume that the lattice $\mathcal{L}$ is normalized such that $\lambda_1(\mathcal{L}) \approx \mathrm{gh}(\mathcal{L}) = 1$. Then under the Gaussian heuristic the list consists of $|L_a| = \alpha^{d+o(d)}$ lattice points, which determines (ignoring nearest neighbour data structures) the space complexity of the algorithm and also determines the time complexity of each iteration.

5.2.2 Average-case CVPP

Recall from Section 2.3 that the approximate γ-CVP problem for $\gamma \geq 1$ asks you, given any target $\mathbf{t}$, to find a lattice point at distance at most $\gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$ from the lattice. Throughout this chapter we will write $c := \gamma^2$ for the squared approximation factor, where $c = 1$ corresponds to exact CVPP. We will only consider the average-case version of this problem, where the lattice follows the Gaussian Heuristic and the target (coset) is uniformly random $\mathbf{t} + \mathcal{L} \sim \mathcal{U}(\mathbb{R}^d/\mathcal{L})$. Under the Gaussian Heuristic this implies that $\mathrm{dist}(\mathcal{L}, \mathbf{t}) \to \mathrm{gh}(\mathcal{L}) = 1$ as $d \to \infty$, and to simplify the analysis we will simply assume that $\mathrm{dist}(\mathcal{L}, \mathbf{t}) = 1$.

The preprocessing variant γ-CVPP additionally allows to do any kind of preprocessing given only a description of the lattice $\mathcal{L}$ (and not the target $\mathbf{t}$). The size of the final preprocessing advice is counted in the eventual space complexity of the CVPP algorithm.

5.2.3 Success probability

The iterative slicer is not guaranteed to succeed as the list does not contain all relevant vectors. However, suppose that the iterative slicer has a success probability of $p_{a,c}$ given a random target. It is clear that having a larger preprocessed list increases the success probability, but in general it is hard to concretely analyse the success probability for a certain list. Heuristically however, we can do such an analysis.

Under the Gaussian heuristic we can derive bounds on $p_{a,c}$, as was first done by [DLW19]. They obtained the following two regimes for the success probability as $d \to \infty$:

- For $a \geq 2c - 2\sqrt{c^2 - c}$ we have $p_{a,c} \to 1$.
- For $a < 2c - 2\sqrt{c^2 - c}$ we have $p_{a,c} = \exp(-C \cdot d + o(d))$ for a constant $C = C(a, c) > 0$ depending on a, c.

The second case above illustrates that for a small list size the algorithm needs to be repeated a large number of times with fresh targets to guarantee a high success probability. This gives us the randomized slicer algorithm. To obtain a fresh target the idea is to sample randomly a not too large element from the coset $\mathbf{t} + \mathcal{L}$, and assume that the reduction of this new target is independent from the initial one. Experiments from [DLW19] suggest that this is a valid assumption to make, and given a success probability $p_{a,c} \ll 1$ it is enough to repeat the algorithm $\Theta(1/p_{a,c})$ times to find the closest lattice point. However this success probability in the case $a < 2c - 2\sqrt{c^2 - c}$ is not yet fully understood. Two heuristic lower bounds [DLW19; Laa19] are known and are shown in Figure 5.3. None of these lower bounds fully dominates the other, which implies that neither of the bounds is sharp. In the remainder of this work we consider this case where we have a small success probability.

5.3 The random walk model

To interpret the iterative slicer algorithm as a random walk we first look at the probability that a target $\mathbf{t}$ is reduced by a random lattice point from the preprocessed list L_a. By the Gaussian heuristic this lattice point is distributed uniformly over the ball of radius $\sqrt{a}$. To

reduce the squared norm $\|\mathbf{t}\|^2$ from x to at most $y \in [(\sqrt{x}-\sqrt{a})^2, x]$ by some $\mathbf{v}$ with $\|\mathbf{v}\|^2 = a$, the inner product between $\mathbf{t}$ and $\mathbf{v}$ must satisfy:

$$\langle \mathbf{t}, \mathbf{v} \rangle \leq -(a+x-y)/2.$$

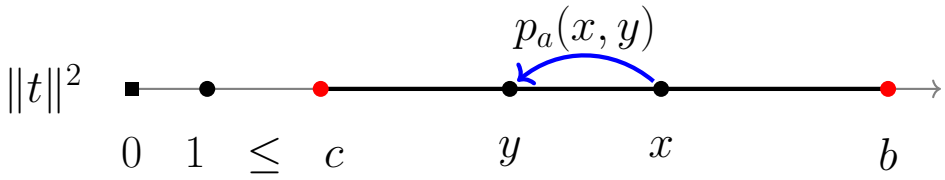
Using the asymtotic formulas in Lemma 46 for the volume of a spherical cap we then deduce the following probability:

$$\Pr_{\mathbf{v} \in \sqrt{a} \cdot \mathcal{B}^d} \left(\|\mathbf{t}+\mathbf{v}\|^2 \leq y \;\middle|\; \|\mathbf{t}\|^2 = x \right) = \left(1 - \frac{(a+x-y)^2}{4ax}\right)^{d/2+o(d)}.$$

Clearly any reduction of $\mathbf{t}$ from a squared norm of x to a squared norm strictly less than $(\sqrt{x}-\sqrt{a})^2$ is unreachable by a vector in $\sqrt{a} \cdot \mathcal{B}^d$. The probability that the target norm is successfully reduced to a value less than $\|t\|^2$ decreases in a and thus we prefer to have short vectors in our list. As the list L_a does not contain just one, but $a^{d/2}$ lattice vectors we obtain the following heuristic reduction probability for a single iteration of the iterative slicer:

$$\begin{aligned} &\Pr\left(\exists \mathbf{v} \in L_a : \|\mathbf{t}+\mathbf{v}\|^2 \leq y \;\middle|\; \|\mathbf{t}\|^2 = x \right)^{2/d} \\ &\quad \to \min\left\{1, a \cdot \left(1 - \frac{(a+x-y)^2}{4ax}\right)\right\} \end{aligned}$$

as $d \to \infty$. The reduction probability takes the form $\exp(-Cd+o(d))$ for some constant $C \geq 0$ that only depends on a, x and y. As we are interested in the limit behaviour as $d \to \infty$ we focus our attention to this base $\exp(-C)$, which we call the base-probability of this reduction and denote it by $p_a(x,y)$. Although these transition probabilities represent a reduction to any square norm $\leq y$, they should asymptotically be interpreted as a reduction to $\approx y$, as for any fixed $\varepsilon > 0$ we have that $p_a(x, y-\epsilon)^d / p_a(x,y)^d = 2^{-\Theta(d)} \to 0$ as $d \to \infty$. If $\|\mathbf{t}\|^2 = x$ is large enough we can almost certainly find a lattice point in L_a that reduces this norm successfully. In fact a simple computation shows that this is the case for any $x > b := a^2/(4a-4)$ as $d \to \infty$. So in our analysis we can assume that our target is already reduced to square norm b, and the interesting part is how probable the remaining reduction from b to c is.



Definition 75 (Transition base-probability). *Let $a, b \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a - 4)$. The transition base-probability $p_a(x, y)$ to reduce $\|\mathbf{t}\|^2$ from x to y is given by*

$$\begin{aligned} p_a : S_a &\to (0, 1], \\ (x, y) &\mapsto \left(a - \frac{(a + x - y)^2}{4x}\right)^{1/2}, \end{aligned}$$

with $S_a = \{(x, y) \in [1, b]^2 : b \geq x \geq y \text{ and } (\sqrt{x} - \sqrt{a})^2 < y\}$ the allowed transitions.

Proof. We have to show that indeed $p_a(x, y) \in (0, 1]$ for $(x, y) \in S_a$, and the given constraints on a and b. For the lower bound note that expanding $(\sqrt{x} - \sqrt{a})^2 < y$, and rewriting gives $a + x - y < 2\sqrt{a}\sqrt{x}$. Because $y \geq x$, and $a > 1$, both sides are positive and thus squaring gives $(a + x - y)^2 < 4ax$, and in particular

$$a - \frac{(a + x - y)^2}{4x} = \frac{4ax - (a + x - y)^2}{4x} > 0.$$

For the upper bound we use that $x \leq b = a^2/(4a - 4)$, and $x - y \geq 0$ to show that

$$4ax - 4x \leq a^2 \leq (a + x - y)^2,$$

from which it follows that

$$\frac{4ax - (a + x - y)^2}{4x} \leq \frac{4x}{4x} = 1.$$

□

Using the above reduction probabilities we model the iterative slicer as a random walk over an infinite graph where the nodes are given by the interval $[1, b]$ such that each node $x_i \in [1, b]$ is associated with the squared norm $\|\mathbf{t}_i\|^2$ of the partly reduced target. Note that

each possible random walk $b = x_0 \to x_1 \to \cdots \to x_n = c$ has a certain success probability. Assuming the different steps are independent this success probability is just the product of the individual reduction probabilities. For an n-step path we could split our list L_a in n parts, one for each step, to obtain this independence without changing the asymptotic size of these lists. Again this success probability is of the form $\exp(-Cd + o(d))$ for some constant $C \geq 0$ that only depends on $x_0, \ldots, x_n$ and a.

Definition 76 (Path). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. Let $n \geq 1$ be an integer. We denote an n-step path on the interval $[1, b]$ by $x_0 \to x_1 \to \cdots \to x_n$ for $x_i \in [1, b]$. We define the set of all n-step paths with positive probability from b to c by:*

$$S_a[b \xrightarrow{n} c] := \{(b = x_0, x_1, \ldots, x_n = c) \in \mathbb{R}^{n+1} : \forall i \ (x_{i-1}, x_i) \in S_a\}.$$

The transition base-probability of such a path is given by

$$P_a[b \xrightarrow{n} c] : S_a[b \xrightarrow{n} c] \to (0, 1],$$
$$\mathbf{x} \mapsto \prod_{i=1}^{n} p_a(x_{i-1}, x_i).$$

The success probability of reaching c from b is determined by the total probability of all successful paths. Note that all these paths have some probability of the form $\exp(-Cd+o(d))$ and thus the probability for the path with the smallest $C \geq 0$ will dominate all other paths for large d. As a result, almost all successful walks will go via the highest probable path, i.e., the one with the highest base-probability. After applying a log-transform this becomes equivalent to finding the shortest path in a weighted graph.

Definition 77 (Transition graph). *Let $a, b \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4)$. Let $V = [1, b]$ and $E = [1, b]^2$ be an infinite graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ given by:*

$$w(x, y) = \begin{cases} -\log p_a(x, y), & \text{if } (x, y) \in S_a; \\ \infty, & \text{otherwise.} \end{cases}$$

One can associate n-step paths in this graph from b to $c \in [1, b)$ with the space $S_a[b \xrightarrow{n} c]$. The length of a path $\mathbf{x} \in S_a[b \xrightarrow{n} c]$ is denoted by $\ell_a[b \xrightarrow{n} c](\mathbf{x})$ and the shortest path length by

$$\ell_{a,opt}[b \to c] = \inf_{n \in \mathbb{Z}_{\geq 1}} \inf_{\mathbf{x} \in S_a[b \xrightarrow{n} c]} \ell_a[b \xrightarrow{n} c](\mathbf{x}).$$

Obtaining the success probability in this model therefore becomes equivalent to obtaining the length of the shortest path $\ell_{a,\mathrm{opt}}[b \to c]$ as we have $P_a[b \xrightarrow{n} c](\mathbf{x}) = \exp(-\ell_a[b \xrightarrow{n} c](\mathbf{x}))$.

5.4 Numerical approximations

We reduced the problem of obtaining the success probability of the iterative slicer to the search of a shortest path in a specially constructed weighted infinite graph. We might not always be able to find an exact solution in the input variables to the length of the shortest path. However for fixed parameters we can always try to numerically approximate the success probability, by approximating the shortest path in our infinite graph. We present two fairly standard methods for doing so. The first method first discretizes the infinite graph and then determines the shortest path using standard algorithms such as Dijkstra's algorithm [Dij59]. The second method uses the fact that the weight function $w_a : S_a \to \mathbb{R}_{\geq 0}$ is convex. A fact that we will later use to derive an explicit solution.

5.4.1 Discretization

A natural way to approximate the shortest path in an infinite graph is to first discretize to a finite subgraph with k vertices. Then one can determine the shortest path in this subgraph using standard methods to obtain a short path in the infinite graph. The details of this approach are shown in Algorithm 6.

Using any optimized Dijkstra implementation the time and space complexity of Algorithm 6 is $O(|E_d| + |V_d| \log |V_d|) = O(k^2)$. In general this method gives a lower bound on the success probability for any fixed a and c. Because the weight function $w_a : S_a \to \mathbb{R}_{\geq 0}$ is continuous Algorithm 6 converges to the optimal path length as $k \to \infty$.

Algorithm 6: A discretized shortest path algorithm

Input: Parameters $1 < a < 2, b = a^2/(4a-4)$ and $c \in [1, b)$ describing the graph, and a discretization value k.

Output: A shortest path on the discretized graph from b to c.

Function `DiscretizedDijkstra(`a, b, c, k`):`

- Compute $V_d = \{c + \frac{i \cdot (b-c)}{k} : i = 0, \ldots, k\}$;
- Compute $E_d = \{(x, y) \in V_d^2 \cap S_a\}$ and the weights $w_a(x, y)$;
- Compute shortest path on $G_d = (V_d, E_d)$ from b to c using [Dij59].

The C++ implementation of this method used for the experiments is made available on Github[2].

For this method to converge to the shortest path in the full graph we only need a continuous weight function. Furthermore the number of steps does not have to be specified a priori. The high memory usage of $O(k^2)$ could limit the fineness of our discretization. To circumvent this we can generate the edges (and their weight) on the fly when needed, which reduces the memory consumption to $O(k)$.

5.4.2 Convex optimization

Where the first method only needed $w_a : S_a \to \mathbb{R}_{\geq 0}$ to be continuous, the second method makes use of the convexity of this function.

Lemma 78 (Convexity of S_a and w_a). *Let $1 < a < 2$, the set of allowed transitions S_a is convex and the weight function w_a is strictly convex on S_a.*

Proof. The convexity of $S_a = \{(x, y) \in [1, b]^2 : b \geq x \geq y \text{ and } (\sqrt{x} - \sqrt{a})^2 < y\}$ follows immediately from the convexity of the function $f(x) = (\sqrt{x} - \sqrt{a})^2$ on $[0, \infty)$. Remember that for $(x, y) \in S_a$

$$w_a(x, y) = -\log p_a(x, y) = -\frac{1}{2} \log \left(a - \frac{(a + x - y)^2}{4x} \right),$$

[2]Implementation: https://github.com/WvanWoerden/randomized-slicer

and thus we have

$$\frac{d^2}{dx^2}w_a(x,y) = \frac{8xp_a(x,y)^2 + (4a - 2(a+x-y))^2 - 16p_a(x,y)^4}{32x^2p_a(x,y)^4},$$
$$\frac{d}{dy}\frac{d}{dx}w_a(x,y) = \frac{-8xp_a(x,y)^2 + (4a - 2(a+x-y)) \cdot 2(a+x-y)}{32x^2p_a(x,y)^4},$$
$$\frac{d^2}{dy^2}w_a(x,y) = \frac{8xp_a(x,y)^2 + 4(a+x-y)^2}{32x^2p_a(x,y)^4}.$$

As $p_a(x,y) > 0$ and $a + x - y \geq a > 0$ for $(x,y) \in S_a$ we have $\frac{d^2}{dy^2}w_a(x,y) > 0$. We consider the Hessian H of w_a. Computing the determinant gives:

$$\det(H) = \frac{2(a+x-y)^4 \cdot (4ax - (a+x-y)^2)}{1024x^6p_a(x,y)^8}$$

and we can conclude that $\det(H) > 0$ from the fact that $4ax - (a + x - y)^2 > 0$ and $(a + x - y)^4 > 0$ for $(x,y) \in S_a$. So H is positive definite, which makes w_a strictly convex on S_a. □

Corollary 79 (Convexity of $S_a[b \xrightarrow{n} c]$ and $\ell_a[b \xrightarrow{n} c]$). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. For any (fixed) integer $n \geq 1$, the space of n-step paths $S_a[b \xrightarrow{n} c]$ is convex and the length function $\ell_a[b \xrightarrow{n} c]$ is strictly convex on $S_a[b \xrightarrow{n} c]$.*

Proof. The convexity of $S_a[b \xrightarrow{n} c]$ follows immediately from that of S_a. Note that $\ell_a[b \xrightarrow{n} c](\mathbf{x}) = \sum_{i=1}^{n} w_a(x_{i-1}, x_i)$ and thus it is convex as a sum of convex functions. Furthermore, for each variable at least one of these functions is strictly convex and thus the sum is strictly convex. □

So for any fixed $n \geq 1$ we can use convex optimization to numerically determine the optimal path of n steps. In fact, because of the strict convexity, we know that this optimal path of n steps (if it exists) is unique. However the question remains what the optimal number of steps is, i.e., for which n we should run the convex optimization algorithm. We might miss the optimal path if we do not guess the optimal number of steps correctly. Luckily because $w_a(b,b) = 0$ by definition, we can increase n without being afraid to skip some optimal path.

Lemma 80 (Longer paths are not worse). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. If $\ell_a[b \xrightarrow{n} c]$ and $\ell_a[b \xrightarrow{n+k} c]$ for $n, k \geq 0$ both attain a minimum, then*

$$\min_{\mathbf{x} \in S_a[b \xrightarrow{n} c]} \ell_a[b \xrightarrow{n} c](\mathbf{x}) \geq \min_{\mathbf{x} \in S_a[b \xrightarrow{n+k} c]} \ell_a[b \xrightarrow{n+k} c](\mathbf{x}).$$

Proof. Suppose $\ell_a[b \xrightarrow{n} c]$ attains its minimum at $\mathbf{y} = (b = y_0, y_1, \ldots, y_n = c) \in S_a[b \xrightarrow{n} c]$. Using that $w_a(b, b) = 0$ we get that:

$$\begin{aligned}\min_{\mathbf{x} \in S_a[b \xrightarrow{n+k} c]} \ell_a[b \xrightarrow{n+k} c](\mathbf{x}) &\leq \ell_a[b \xrightarrow{n+k} c](b, \ldots, b = y_0, \ldots, y_n = c) \\ &= k \cdot w_a(b, b) + \ell_a[b \xrightarrow{n} c](\mathbf{y}) \\ &= \ell_a[b \xrightarrow{n} c](\mathbf{y}).\end{aligned}$$

This completes the proof. □

So increasing n can only improve the optimal result. When running a numerical convex optimization algorithm one could start with a somewhat small n and increase it (e.g., double it) until the result does not improve any more.

5.4.3 Numerical results

We ran both numerical algorithms and got similar results. Running the convex optimization algorithm gave better results for small $a = 1+\varepsilon$ as the fineness of the discretization is not enough to represent the almost shortest paths in this regime. This is easily explained as $b \approx \frac{1}{4\varepsilon}$ and thus for fixed c the distance between b and c, i.e., the interval to be covered by the discretization quickly grows as $\varepsilon \to 0$.

The new lower bound that we obtained numerically for exact CVPP ($c = 1$) is shown in Figure 5.3. For $\alpha \leq 1.1047$ we observe that the new lower bound is strictly better than the two previous lower bounds. For $\alpha > 1.1047$ the new lower bound is identical to the lower bound from [DLW19]. Taking a closer look at the short paths we obtained numerically we see that $\alpha \approx 1.1047$ is exactly the moment where this path switches from a single step to at least 2 steps. This makes sense as in our model the lower bound from [DLW19] can be interpreted as

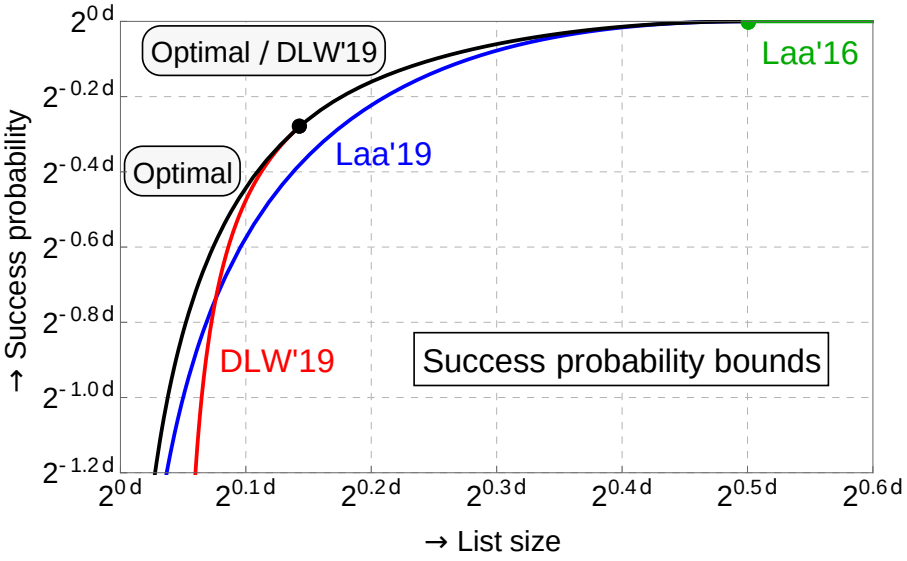


Figure 5.3: Lower bounds on success probability of the iterative slicer for CVPP ($c = 1$) computed with a discretization parameter of $k = 5000$.

a 'single step' analysis. This also explains the asymptote for this lower bound as for $\alpha \leq 1.0340$ it is not possible to walk from b to $c = 1$ in a single step.

When inspecting these short paths $b = x_0 \to x_1 \to \cdots \to x_n = c$ further we observed an almost perfect fit with a quadratic formula $x_i = u \cdot i^2 + v \cdot i + b$ for some constants u, v. In the next section we show how we use this to obtain an exact analytic solution for the shortest path.

5.5 An exact solution

In order to determine an exact solution of the shortest path, and thus an exact solution of the success probability of the iterative slicer we use some observations from the numerical results. Due to Corollary 79 we know that for any fixed $n \geq 1$ our minimization problem is strictly convex. As a result there can be at most one local minimum which, if it exists, is immediately also the unique global minimum.

What remains is to explicitly construct such a local minimum, after which we only have to optimize over n. We recall from Section 5.4.3 that the optimal path $x_0 \to \cdots \to x_n$ seems to take the shape $x_i = u \cdot i^2 + v \cdot i + b$ with $x_n = c$. So for our construction we assume this shape, which reduces the problem to determining the constants u, v. Furthermore, as we are trying to construct a local minimum, we assume that all partial derivatives in the non-constant variables are equal to 0. This gives enough restrictions to obtain a verbose, but explicit solution.

Definition 81 (Explicit construction). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$, and let $n \geq 1$ be an integer. We define*

$$x_i := u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b,$$

with $u_a[b \xrightarrow{1} c] := 0$, $v_a[b \xrightarrow{1} c] := c - b$ and for $n \geq 2$:

$$u_a[b \xrightarrow{n} c] := \frac{(b+c-a)n - \sqrt{(an^2-(b+c))^2 + 4bc(n^2-1)}}{n^3-n},$$
$$v_a[b \xrightarrow{n} c] := \frac{(a-2b)n^2 + (b-c) + \sqrt{(an^2-(b+c))^2 + 4bc(n^2-1)}n}{n^3-n}.$$

Lemma 82. *Let a, b, c, n and x_i be as in Definition 81. By construction we have $x_n = c$ and*

$$\frac{\partial}{\partial x_i} \sum_{j=1}^{n} -\log p_a(x_{j-1}, x_j) = 0$$

for all $i \in \{1, \ldots, n-1\}$.

Proof. Note that the partial derivative constraints can be reduced to the single constraint $\frac{\partial}{\partial x_i}(-\log p_a(x_{i-1}, x_i) - \log p_a(x_i, x_{i+1})) = 0$ for a symbolic i. Together with the constraint $x_n = c$ one can solve for u, v in $x_i = u \cdot i^2 + v \cdot i + b$. For a symbolic verification see the Sage script in Appendix A of the full version[3]. □

What remains is to show that the explicit construction indeed gives a valid path, i.e., one that is in the domain $S_a[b \xrightarrow{n} c]$. An example of

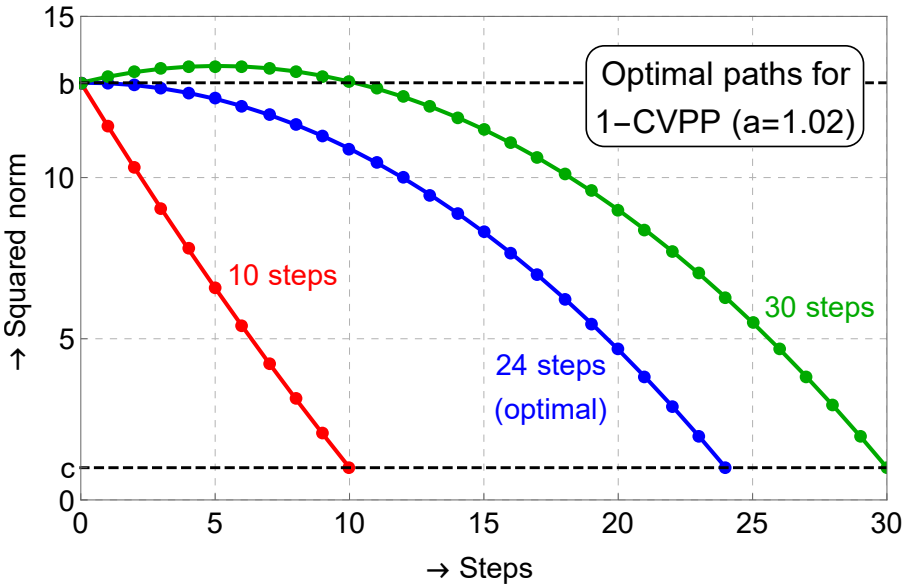


Figure 5.4: Some examples of the constructed paths in Definition 81 for $a = 1.02, c = 1$.

how these constructed paths look are given in Figure 5.4. We observe that if n becomes too large these constructed paths are invalid as they walk outside the interval $[c, b]$. This is an artefact of our simplification that $w_a(x, y) = -\log p_a(x, y)$ which does not hold for $(x, y) \notin S_a$. We can still ask the question for which n this construction is actually valid.

Lemma 83 (Valid constructions). *Let a, b, c be as in Definition 81. Let $n \geq 1$ be an integer such that $\frac{b-c}{a} \leq n < \frac{1}{2} + \frac{\sqrt{(4b-a)^2 - 8(2b-a)c}}{2a}$ and*

$$x_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b.$$

Then $\mathbf{x} = (x_0, \ldots, x_n) \in S_a[b \xrightarrow{n} c]$ and $\mathbf{x}$ is the unique minimum of $\ell_a[b \xrightarrow{n} c]$.

Proof. We have to check that $\mathbf{x}$ satisfies the two conditions

$$x_{i-1} \geq x_i \qquad \text{and} \qquad (\sqrt{x_{i-1}} - \sqrt{a})^2 < x_i,$$

[3]Full version: https://eprint.iacr.org/2020/120.pdf

for all $i \in \{1, \dots, n\}$. Note that for $n = 0$ we must have $b = c$ and the statement becomes trivial. For $n = 1$ we have $\mathbf{x} = (b, c)$ and the conditions follows from $0 \le b - c \le na \le a$. So we can assume that $n \ge 2$. First we rewrite $u_a[b \overset{n}{\to} c]$ to:

$$u_a[b \overset{n}{\to} c] = \frac{(b+c-a)n - \sqrt{((b+c-a)n)^2 + (a^2n^2 - (b-c)^2)(n^2-1)}}{n^3 - n},$$

which makes it clear that $u_a[b \overset{n}{\to} c] \le 0$ when $an \ge b - c$. As a result the differences

$$x_{i-1} - x_i = (1 - 2i) \cdot u_a[b \overset{n}{\to} c] - v_a[b \overset{n}{\to} c],$$

are increasing in $i \in \{1, \dots, n\}$. Therefore for the first condition it is enough to check that

$$x_0 - x_1 = \frac{(b-c) + (2b-a)n - \sqrt{(an^2 - (b+c))^2 + 4bc(n^2-1)}}{n^2 + n} \ge 0.$$

In fact a solution with $x_0 = x_1 = b$ is not so interesting, so solving for $x_0 - x_1 > 0$ gives for $n \ge 2$ the sufficient condition

$$n < \frac{1}{2} + \frac{\sqrt{(4b-a)^2 - 8(2b-a)c}}{2\,a}.$$

For the second condition we first show the stronger property that $x_{i-1} - x_i \le a$, and again by the increasing differences it is enough to show that $x_{n-1} - x_n \le a$; rewriting gives the following sufficient statement for $n \ge 2$:

$$-an + b - c \le 0.$$

Now we prove that $\sqrt{x_{i-1}} - \sqrt{x_i} < \sqrt{a}$. If $x_{i-1} = x_i$ the condition holds trivially, else $x_{i-1} > x_i$ and we get

$$(\sqrt{x_{i-1}} - \sqrt{x_i})^2 < (\sqrt{x_{i-1}} - \sqrt{x_i})(\sqrt{x_{i-1}} + \sqrt{x_i}) = x_{i-1} - x_i \le a.$$

Note that if $\sqrt{x_{i-1}} - \sqrt{a} \ge 0$, then we can rewrite $\sqrt{x_{i-1}} - \sqrt{x_i} < \sqrt{a}$ and square to obtain the condition $(\sqrt{x_{i-1}} - \sqrt{a})^2 < x_i$. Alternatively, if $\sqrt{x_{i-1}} - \sqrt{a} < 0$, then due to $a < 2$ we obtain $(\sqrt{x_{i-1}} - \sqrt{a})^2 < (1 - \sqrt{2})^2 < 1 \le x_i$.

We conclude that $\mathbf{x} \in S_a[b \overset{n}{\to} c]$, and because we have the equality $\ell_a[b \overset{n}{\to} c](\mathbf{x}) = \sum_{i=1}^{n} -\log p_a(x_{i-1}, x_i)$ on $S_a[b \overset{n}{\to} c]$, the claim that this is a global minimum follows from Definition 81 and Lemma 79. □

So by Lemma 81 there exists some $s \in \mathbb{N}$ such that for all $(b - c)/a \leq n \leq s$ we have an explicit construction for the optimal n-step path. By Lemma 80 we know that of these paths the one with $n = s$ steps must be the shortest. However for $n > s$ our construction did not work and thus we do not know if any shorter path exists. Inspired by Lemma 80 and numerical results we obtain the following alternative exact solution for $n > s$.

Theorem 84 (Optimal arbitrary-step paths). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a - 4) > c \geq 1$. Let n satisfy*

$$n = \left\lceil -\frac{1}{2} + \frac{1}{2a}\sqrt{(4b - a)^2 - 8(2b - a)c} \right\rceil . \tag{5.1}$$

For $k \geq n$ the unique global minimum of $\ell_a[b \xrightarrow{k} c]$ is given by

$$\mathbf{x} = (b, \ldots, b, b = y_0, \ldots, y_n = c) \in S_a[b \xrightarrow{k} c]$$

with $y_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b$ and the length is equal to $\ell_a[b \xrightarrow{n} c](\boldsymbol{y})$.

Proof. By Corollary 79 it is enough to show that $\mathbf{x}$ is a local minimum, therefore we check the partial derivatives. For $i > k - n$ we have $\frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) = \frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{n} c](\mathbf{y}) = 0$ by construction. For $i < k - n$ we have $x_{i-1} = x_i = x_{i+1} = b$, which results in $\frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) = -\frac{a-1}{2b} < 0$. For the most interesting case $i = k - n$ we need that $n \geq -\frac{1}{2} + \frac{\sqrt{(4b-a)^2-8(2b-a)c}}{2\,a}$. Because as a result we get $y_0 - y_1 \leq \frac{a^2}{2b-a}$, which together with $y_0 - y_1 \leq b - c \leq b - 1$ is precisely enough to show that $\frac{\partial}{\partial x_{k-n}}\ell_a[b \xrightarrow{k} c](\mathbf{x}) \leq 0$.

To conclude let $\boldsymbol{z} \neq \mathbf{x} \in S_a[b \xrightarrow{n} c]$, then by Corollary 79 and using that $z_i - x_i = z_i - b \leq 0$ for all $0 \leq i \leq k - n$ we have:

$$\begin{aligned}
\ell_a[b \xrightarrow{k} c](\boldsymbol{z}) &> \ell_a[b \xrightarrow{k} c](\mathbf{x}) + \langle \boldsymbol{y} - \mathbf{x}, \nabla \ell_a[b \xrightarrow{k} c](\mathbf{x}) \rangle \\
&= \ell_a[b \xrightarrow{k} c](\mathbf{x}) + \sum_{i \leq k-n} (z_i - x_i) \cdot \frac{\partial}{\partial x_i}\ell_a[b \xrightarrow{k} c](\mathbf{x}) \\
&\geq \ell_a[b \xrightarrow{k} c](\mathbf{x}).
\end{aligned}$$

and thus $\mathbf{x}$ is the unique global minimum of $\ell_a[b \xrightarrow{k} c]$. □

Corollary 85 (Optimal minimum-step paths). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. The optimal path in the transition graph in Definition 77 from b to c consists of n steps, with n defined by equation (5.1). The optimal path is of the form $b = x_0 \to x_1 \to \cdots \to x_n = c$ with $x_i = u_a[b \xrightarrow{n} c] \cdot i^2 + v_a[b \xrightarrow{n} c] \cdot i + b$.*

Heuristic claim 86. *Given the optimal path $b = x_0 \to \cdots \to x_n = c$ from Corollary 85, the success probability of the iterative slice algorithm with list size $a^{n/2+o(n)}$ for $\sqrt{c}$-CVPP is given by*

$$\exp\left(-\sum_{i=1}^{n} w_a(x_{i-1}, x_i)d + o(d)\right).$$

As we have an exact formula for the optimal number of steps, and the lower bound from [DLW19] uses a 'single-step' analysis we know exactly in which regime Corollary 85 improves on theirs. Namely for those $a > 1$ and $c \geq 1$ such that for n defined by equation (5.1) we have $n > 1$. For exact CVPP we obtain improvements for $a < 1.22033$, i.e., when using less than $2^{0.1436d+o(d)}$ memory, as can be seen in Figure 5.1. This improvement can also be visualized through Figure 5.5, which plots the optimal number of steps against the size of the preprocessed list. Whenever the optimal strategy involves taking more than one step, we improve upon [DLW19]. For the crossover points where the number of optimal steps changes we have a more succinct formula for the shortest path and the success probability.

Lemma 87 (Success probability for integral n). *Let $a, b, c \in \mathbb{R}$ be such that $1 < a < 2$ and $b = a^2/(4a-4) > c \geq 1$. If n defined similar to equation (5.1), but without rounding up, is integral, then the optimal path from b to c has probability*

$$\left(\left(\frac{a}{2-a}\right)^n \cdot \left(1 - \frac{2n(a-1)}{2-a}\right)\right)^{d/2+o(d)}.$$

Proof. For such n we obtain the expression $x_i = b - (i+1) \cdot i \cdot \frac{a^2-a}{2-a}$. The result follows from simplifying the remaining expression. □

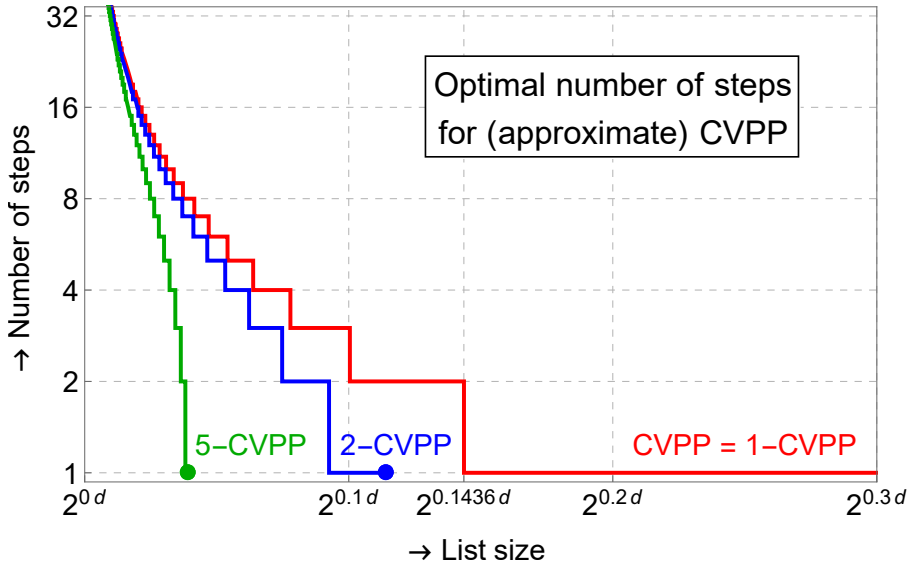


Figure 5.5: Optimal number of steps n against the list size $|L| = \alpha^{d+o(d)} = a^{d/2+o(d)}$. We improve upon [DLW19] whenever $n > 1$. For large list sizes the optimal number of steps of cost $\exp(-Cd + o(d))$ drops to 0, as then the success probability of the iterative slicer equals $2^{-o(d)}$.