Generating information-dense promoter sequences with optimal string packing

Virgile Andreani[1,3+], Eric J. South[2,3+], Mary J. Dunlop[1,2,3*]

[1] Biomedical Engineering Department, Boston University, Boston, MA 02215
[2] Molecular Biology, Cell Biology & Biochemistry Program, Boston University, Boston, MA 02215
[3] Biological Design Center, Boston University, Boston, MA 02215
[+] These authors contributed equally to this work.
* Corresponding Author: mjdunlop@bu.edu

Short title: Generating dense promoter sequences with optimal string packing

Abstract

Dense arrangements of binding sites within nucleotide sequences can collectively influence downstream transcription rates or initiate biomolecular interactions. For example, natural promoter regions can harbor many overlapping transcription factor binding sites that influence the rate of transcription initiation. Despite the prevalence of overlapping binding sites in nature, rapid design of nucleotide sequences with many overlapping sites remains a challenge. Here, we show that this is an NP-hard problem, coined here as the nucleotide String Packing Problem (SPP). We then introduce a computational technique that efficiently assembles sets of DNA-protein binding sites into dense, contiguous stretches of double-stranded DNA. For the efficient design of nucleotide sequences spanning hundreds of base pairs, we reduce the SPP to an Orienteering Problem with integer distances, and then leverage modern integer linear programming solvers. Our method optimally packs libraries of 20-100 binding sites into dense nucleotide arrays of 50-300 base pairs in 0.05-10 seconds. Unlike approximation algorithms or meta-heuristics, our approach finds provably optimal solutions. We demonstrate how our method can generate large sets of diverse sequences suitable for library generation, where the frequency of binding site usage across the returned sequences can be controlled by modulating the objective function. As an example, we then show how adding additional constraints, like the inclusion of sequence elements with fixed positions, allows for the design of bacterial promoters. The nucleotide string packing approach we present can accelerate the design of sequences with complex DNA-protein interactions. When used in combination with synthesis and high-throughput screening, this design strategy could help interrogate how complex binding site arrangements impact either gene expression or biomolecular mechanisms in varied cellular contexts.

Author Summary

The way protein binding sites are arranged on DNA can control the regulation and transcription of downstream genes. Areas with a high concentration of binding sites can enable complex interplay between transcription factors, a feature that is exploited by natural promoters. However, designing synthetic promoters that contain dense arrangements of binding sites is a challenge. The task involves overlapping many binding sites, each typically about 10 nucleotides long, within a constrained sequence area, which becomes increasingly difficult as sequence length decreases, and binding site variety increases. We introduce an approach to design nucleotide sequences with optimally packed protein binding sites, which we call the nucleotide String Packing Problem (SPP). We show that the SPP can be solved efficiently using integer linear programming to identify the densest arrangements of binding sites for a specified sequence length. We show how adding additional constraints, like the inclusion of sequence elements with fixed positions, allows for the design of bacterial promoters. The presented approach enables the rapid design and study of nucleotide sequences with complex, dense binding site architectures.

Introduction

The layout of binding sites within nucleotide sequences can define both biomolecular interactions and subsequent biological processes. For example, the architecture of cis-regulatory regions—stretches of non-coding DNA containing binding sites for transcription factors and other regulatory proteins—plays a key role in controlling the expression of downstream genes. These regions serve to recruit regulators and convert cellular signals into transcriptional outputs (1–4). Regions with few binding sites are thought to enable transcription factors to bind and then permit higher overall transcription levels, even when these proteins are in limited quantities (5). Conversely, dense clusters of binding sites can elicit complex phenomena such as cooperative binding effects (6–9), steric hindrance (10,11), and transcription factor sharing (5). These emergent properties can lead to nonlinear transcriptional regulatory logic, wide dynamic ranges, and context-sensitive gene expression (12–14).

As the complexity of cis-regulatory regions becomes more evident, researchers have begun to employ forward engineering approaches to systematically map promoter sequences to transcriptional readouts. One of the most prominent methods in this regard is the use of massively parallel reporter assays (MPRAs) (15–17). These assays link the expression of a reporter gene to a specific cis-regulatory variant, which is often situated upstream on either an episomal or genomically integrated locus (18). Traditionally, the design of promoter variant libraries for MPRAs has followed one of two different strategies: either diffuse nucleotide diversification, achieved through methods like error-prone PCR or random mutagenesis (19,20), or hybrid engineering approaches that fuse core promoter subregions with other discrete binding site elements (21,22). In hybrid engineering approaches, the design of sequence variants for MPRAs has typically focused on adjusting binding site spacing and consensus sequences, often placed in adjacent or proximal positions, while overlooking the potential for overlapping binding sites (15,23–26). While these studies offer insights into how binding site positioning and biophysical constraints inform promoter strength, their general omission of overlapping binding sites has limited the characterization of nonlinear, emergent properties that can arise in natural densely arranged cis-regulatory regions.

Designing nucleotide sequences with overlapping binding sites becomes challenging when the total length of the desired binding sites, each typically about 10 nucleotides long (27), surpasses the fixed length of the intended output sequence. The overall problem increases in complexity as the pool of binding sites expands, causing an exponential increase in potential binding site configurations. Studies that have tackled the design of overlapping sequences, and their effects on transcriptional regulatory logic, often rely on *ad hoc* methods or generate a limited set of short sequences (26,28–30), thus restricting the generation of large libraries. Meanwhile, generative AI techniques are starting to show promise in emulating the complexity of context-dependent promoters (31–37). However, these models often struggle with interpretability, and fine-tuning them to include or exclude specific binding sites still requires specialized expertise (38).

Here we present a novel computational method for the design of nucleotide sequences with densely packed DNA-protein binding sites, which we name the nucleotide String Packing Problem (SPP), related to the classical Shortest Common Superstring problem in theoretical computer science (39). We deliberately use the terminology "string" instead of "sequence" in the problem name to avoid confusion with the concept from computer science of a "subsequence," which is not necessarily contiguous. This distinction is crucial for distinguishing the SPP, where an individual binding site must not be split, from a potential "Sequence Packing Problem" where the sequence AGGA of length 4 would fit the three elements AA, GG, AGA (among others), due to the non-contiguous nature of subsequences.

After proving that the SPP is NP-hard, we reduce it to the Orienteering Problem with integer distances, which is an optimization problem related to the Traveling Salesman Problem (40). This reduction allows us to formulate nucleotide string packing as an integer linear programming problem, which can then be solved efficiently with a variety of open-source and commercial solvers. This formulation not only

underscores the computational complexity of the design task but also lays the groundwork for generating sequences that accommodate a maximal number of protein binding sites. The expressivity of integer linear programming also allows for a variety of design objectives and constraints, which can be used to tailor the specifics of the output nucleotide sequence. We demonstrate how a simple change to the model makes it possible to adjust the order of solutions returned to favor a more uniform representation of binding sites across sequences. We also demonstrate that by adding additional constraints to the SPP, like the inclusion of sequence elements with fixed positions, we can effectively interweave dense regions of binding sites around -35 and -10 sigma factor recognition sites. This approach enables the design of complex promoters that interact with multiple sigma factors. The integer linear programming model and all its extensions have been made available to the community as part of an open-source Python library available at https://gitlab.com/dunloplab/dense-arrays. Equally applicable to large-scale libraries and more focused studies, this computational approach serves as a valuable resource for designing promoters with complex DNA-protein binding architectures.

Results

Complexity of the String Packing Problem (SPP)

The problem of packing DNA-protein binding sites into a DNA sequence of a fixed length can be formulated more generally as an optimization problem on strings: consider a finite alphabet Σ, a library R of strings from Σ^*, and a natural integer L. The goal is to find a string w of length L that maximizes the number of different strings of R contained in w, that is, the number of strings $x \in R$ such that there exists $w_0 \in \Sigma^*$ and $w_1 \in \Sigma^*$ such that $w = w_0 x w_1$ (Figure 1A).

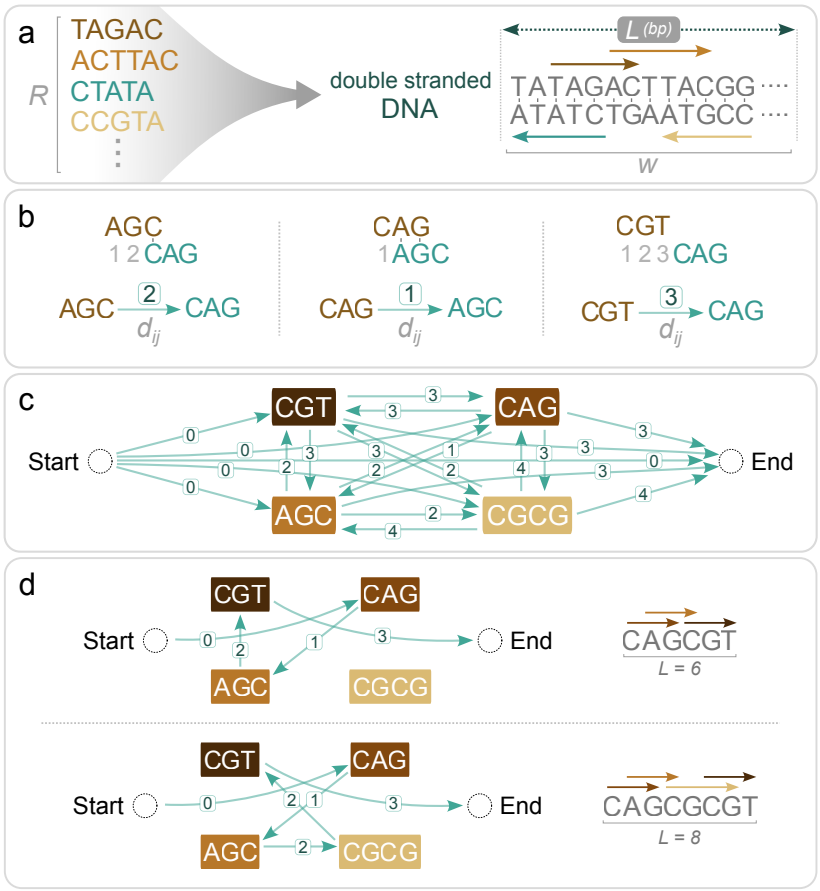

Figure 1. Formulation of the nucleotide String Packing Problem (SPP) as an Orienteering Problem (OP). (A) The SPP consists of fitting as many different strings from a library as possible inside a string of a given length L. bp,

base pairs. (B) We define an asymmetrical metric between strings i and j, d_{ij}, as the least number of shifts needed for the prefix of the second string to match the suffix of the first string. (C) After adding Start and End vertices, we construct the complete graph between library strings with the associated shift, d_{ij}, between strings shown. The OP is solved on this graph. For the double-stranded problem, we also include the reverse-complements of all the binding sites, omitted here for clarity. (D) Representative solutions for two different values of L.

To determine the complexity of this problem, let us first consider the associated decision problem (SPP-DECISION): given the alphabet Σ, the library R, two natural integers L and N, is there a string of length L or fewer that contains N or more of the strings of R? We will show that this problem is NP-complete, that is, it belongs to the set of NP problems at least as hard as all other NP problems.

To show that SPP-DECISION is NP-complete, we first need to show that it is an NP problem, and then that it is at least as hard as all other NP problems. (1) To show that it is an NP problem, we need to show that we can construct certificates of positive answers, with sizes that are polynomial in the size of the input, which can then be verified by a polynomial-time algorithm. (2) To show that SPP-DECISION is at least as hard as all other NP problems, we will proceed by reduction from another NP-complete problem: the decision problem corresponding to the Shortest Common Superstring problem (SCS-DECISION) (41) (as described in (39)). The proof follows:

(1) SPP-DECISION is in NP: A certificate of existence of a superstring can be a list of the strings included in it with their offsets from the beginning of the superstring. The size of this certificate is polynomial in the size of the problem input. Moreover, this certificate can be verified by a polynomial-time algorithm. SPP-DECISION is then in NP.
(2) SCS-DECISION is reducible to SPP-DECISION: The SCS-DECISION problem is closely related to the SPP-DECISION problem, in that given Σ, R, and L, the question is to determine if there exists a string of length L or fewer that contains all strings from R. Considering any SCS-DECISION problem described by (Σ, R, L), we can build an SPP-DECISION problem described by the same alphabet Σ, library R, and integer L, and where $N = |R|$. A positive (respectively negative) answer to this SPP-DECISION problem implies a positive (respectively negative) answer to the original SCS-DECISION problem, which shows that SCS-DECISION is reducible to SPP-DECISION, and thus that SPP-DECISION is NP-complete.

Let us now consider the original optimization problem, SPP, which is not a decision problem. We will show that it is NP-hard, that is, every problem in NP is reducible to it. To show this, we will show that SPP-DECISION, which we just showed is NP-complete, is reducible to the SPP. Let us consider any SPP-DECISION problem, described by (Σ, R, L, N). We can construct the SPP described by (Σ, R, L). Let us call M the number of different substrings of R included in the optimal solution to this problem. It is enough to compare M to N to answer the SPP-DECISION problem, thus showing that SPP-DECISION is reducible to SPP and therefore that SPP is NP-hard.

Performance and scalability

Brute force approaches to solving the SPP are not scalable, due to the NP-hardness of the task. No polynomial-time algorithms are known to solve it optimally, and the only deterministic algorithms known all eventually rely on an exponential number of steps in the worst case. But the choice of the approach greatly impacts the practical solve time. For example, for a library with $|R| = 50$ binding sites and a sequence of length $L = 200$ base pairs, one brute force approach could be to enumerate all possible $4^{200} \approx 10^{120}$ sequences, which is computationally infeasible. Despite their computational challenges, many NP-hard problems hold significant practical value. This has led to the development of specialized solvers designed to efficiently manage these problems, although they may face difficulties with extremely large instances (43). Many numerical solvers have been developed to tackle the NP-hard integer linear programming problem, which restricts linear programming to integer variables. We formulated the SPP as an integer

linear programming problem by reducing it to a variant of the Travelling Salesman Problem (TSP) known as the Orienteering Problem (OP) (40,44,45) (see Methods for the integer linear model). This formulation abstracts away the notion of nucleotides and considers instead the interaction between binding sites through a shifting metric defined between every pair of binding sites (Figure 1B): this leads to a dramatic decrease in the complexity of the problem. Indeed, a brute force approach to solve this formulation would now only need to enumerate all possible paths in the graph visiting each binding site at most once (Figure 1C-D). For the library considered above ($|R| = 50$ and $L = 200$), there are about 10^{65} such paths (as calculated using the approach in Ref. (42)), which is still astronomically large, but much less than 10^{120}. Thus, the same problem can be formulated in different ways, and the amount of computational work needed to solve them can be very different. The algorithms in the integer linear programming are also exponential but are designed to dramatically decrease the exponential constant. While they cannot cheat the exponential for very large tasks, they embed heuristics that help them to quickly find an optimal solution in many cases. Here, thanks to their internal algorithms and heuristics, the integer linear programming solvers do not have to explore all 10^{65} paths of the OP, and thus are able to find a solution for a problem of size $|R| = 50$ and $L = 200$ highly efficiently, returning solutions within seconds. The strength of our approach relies on two factors: the formulation of the initial SPP as a graph problem abstracts the notion of nucleotides and eliminates the need to represent them individually, which drastically cuts down the search space, and the use of an optimized solver further reduces the complexity by tackling this problem using state of the art algorithms.

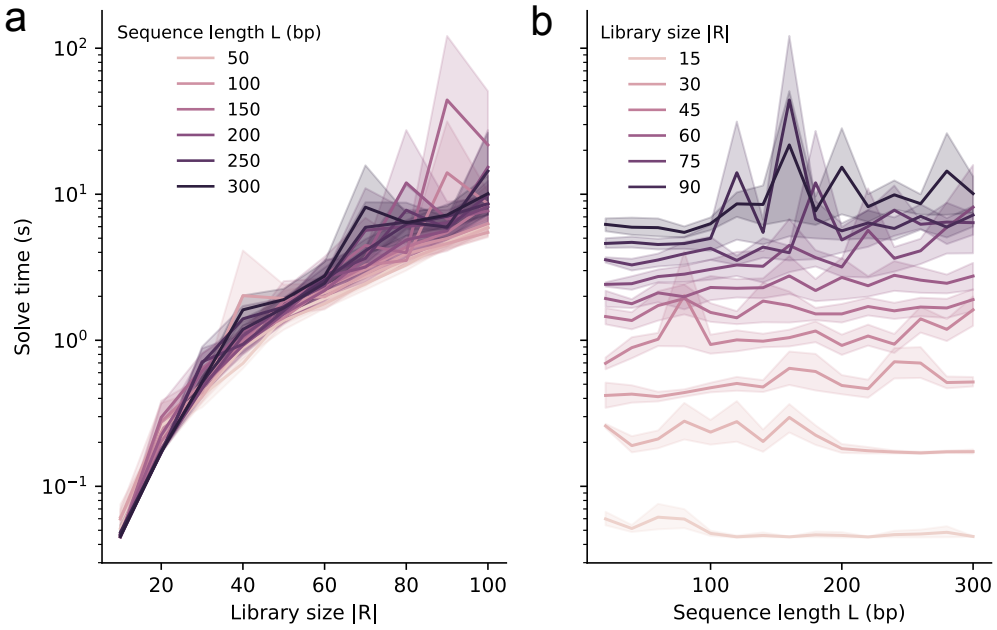

Figure 2. Performance of the SPP. Solve time as a function of (A) library size, $|R|$, and (B) sequence length, L. All experiments were run 10 times, with the Gurobi v10.0.1 backend. The shaded regions represent the bootstrapped 95% confidence interval around the mean. All replicates use a different, randomly sampled library, with random binding sites of uniformly random length between 5 and 15 base pairs. The library size and sequence length are specified on the graphs. Double-strand optimization is performed here. For all these experiments, Gurobi was given access to an Intel Xeon E5-2650 v2 CPU (16 logical threads) and its memory usage was under 3 GB. A timeout of 600 seconds was set, reached only 8 times out of 1500, which we excluded from these data.

Solve times for the SPP are expected to be dependent on the parameters of the problem (size of the library, $|R|$, and length of the sequence, L). To investigate the influence of these parameters, we considered library sizes, $|R|$, with between 10-100 distinct binding sites, each with a random length between 5 and 15 base pairs, where these lengths are based on known protein-DNA interactions (27). We also considered typical

sizes of cis-regulatory regions between 20-300 base pairs (46) to use as our sequence length, L. Using the highly efficient Gurobi solver as a backend for the integer linear programming, we found that the solve times were rapid (Figure 2A). For example, the library with $|R| = 50$ binding sites and a sequence length of $L = 200$ base pairs can be solved in 2 seconds on a laptop. We found that solve time scales proportionally with the number of variables of the model, that is, with the square of the library size. The solve time is also largely independent of the length of the sequence (Figure 2B), which may be explained by the fact that this parameter does not change the number of variables or the structure of the model. It is also possible that increasing the length of the sequence may increase the number of optimal solutions, making it easier for the solver to find one of them. The solve times range from fractions of seconds for libraries of size $|R| = 20$, to 10 seconds for libraries of size 100, making this is an efficient and accessible method for realistic nucleotide design tasks.

Heuristic approaches to solve the problem

In some cases, it is not necessary to reach the optimal solution to an NP-hard problem if a suboptimal but sufficiently good solution is acceptable. It can be much easier to find suboptimal solutions, and this can often be achieved by "approximation algorithms," many of which run in polynomial time. For instance, in the case of the TSP, the nearest neighbor algorithm—always visiting the closest city not yet visited—operates in time proportional to the square of the number of cities, yet it may yield solutions significantly worse than the optimal. For metric TSPs (i.e., when the distance respects the triangle inequality), the Christofides – Serdyukov algorithm is a polynomial-time algorithm that is guaranteed to return a solution no longer than 1.5 times the optimal solution (47–49). Approximation algorithms also exist for the SCC problem (50). The greedy algorithm (repetitively merging the two strings with the best overlap, until there is only one remaining) will return a superstring at most 3.425 times longer than the optimal string. A recent approximation algorithm allows this factor to be lowered to 2.475 (51). Thus, if applied to our problem, a similar strategy would create an array in the worst case less than half as dense as it could be.

To test if an approximation method could produce results comparable to the optimal results returned by our integer linear programming approach, we employed a greedy algorithm. This algorithm involves repetitively merging the two binding sites with the best overlap until only one is left (in the worst case this is 3.425 times longer than the optimal solution). If this resulting sequence is shorter than the targeted sequence length, while fitting all the library binding sites, it is an optimal solution and is thus returned. If the sequence extends beyond the target length, we apply a sliding window corresponding to our target length, and then locate the sequence segment containing the highest number of binding sites; this segment is subsequently returned. In a numerical study using randomly generated binding sites, we compared the number of binding sites in the solutions returned by this greedy algorithm to the optimal number computed with our integer linear programming formulation, finding that the approximation algorithm often returned fewer binding sites than optimal (60% of the optimal in the worst cases in our numerical studies), with large library sizes ($|R|$) and compact sequence lengths (L) posing challenges that only the integer linear programming formulation was able to tackle (Figure S1). Meta-heuristics—general strategies applicable to a range of problems—are an alternative approach and have shown promise in identifying quality suboptimal solutions. Genetic algorithms, for instance, have shown efficacy in similar nucleotide overlapping problems (28,29). However, the practical application of these algorithms is limited by difficulties in tuning them to yield good solutions quickly.

Diversity of the generated solutions

Next, we returned to our integer linear programming approach and quantified the diversity of sequence solutions returned by the algorithm. Some applications might require the generation of multiple sequences, composed of binding sites from the same library, for the purposes of library curation or biological screening. For example, generating diverse DNA sequences from the same binding site library can enable nuanced studies in functional genomics, or facilitate comparative analysis during either drug screens or synthetic

biology circuit optimizations.

We found that the integer linear programming approach introduces a bias for smaller binding sites because of the target to maximize the number of sites in a sequence (Figure S2). Interestingly, we found that this bias is not exclusively tied to size variation, as it manifests even among equally sized binding sites. To demonstrate this, we created random libraries of $|R| = 10$ binding sites of 10 base pairs each. We enumerated the best solutions for a final sequence of length $L = 50$ base pairs: in all cases, six binding sites were fit in this sequence at best (score = 6) (Figure 3A). For the three random libraries we tested, the number of different sequences realizing this best score were 1854, 2654, and 10,922, where the higher numbers correspond to a case where there were more ways to create sequences with 6 binding sites, for example due to similar sequences in the starting library. If the frequency of binding site inclusion were perfectly uniform, we would expect each binding site to occur with a frequency of 0.1 ($1/|R|$). However, we found that every binding site was not equally likely to be part of these best solutions (Figure 3B). To rule out the chance that these findings were simply due to sampling variations, we replicated the procedure, randomly selecting six binding sites from the library as many times as there were solutions, without attempting to fit them into a sequence. To compare this sampling bias between these two scenarios, we sorted binding sites by decreasing frequency (Figure 3B-3C). The flatter the line, the more uniform the distribution is. Conversely, a steeper slope indicates a less diverse distribution, tending to favor certain binding sites over others. We found that the creation of dense arrays inherently tends to favor specific binding sites, shaped by the unique interplay among sites within each specific library (Figure 3D).

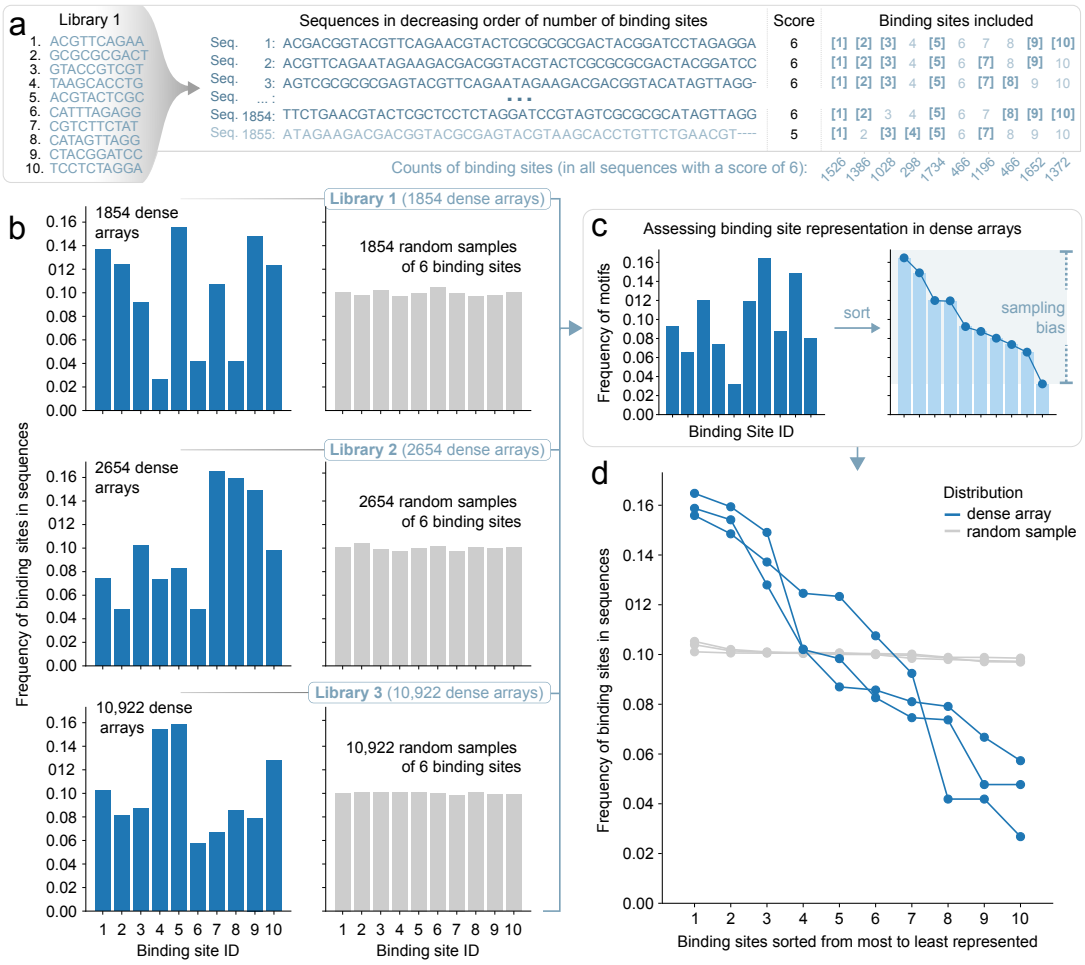

Figure 3. Assessing binding site representation across dense arrays. (A) For the $|R| = 10$ randomly generated

binding sites of Library 1 we found 1854 optimal sequences (each fitting 6 binding sites within L = 50 bp). We then tallied up the binding sites involved in these sequences. (B) The frequency of binding site distribution is represented as a histogram. Three random libraries of $|R|$ = 10 binding sites, each of 10 base pairs in length, were used to create sequences of length L = 50 bp. The bar plots show the frequency at which each of the 10 binding sites was present in these sequences. Optimal solutions using the dense array approach with the SPP (blue) and using random samples of 6 binding sites (gray) are shown. (C) Sorting the histogram gives an idea of the heterogeneity of the distribution, where larger differences between the frequency of binding site usage correspond to larger sampling biases. (D) The histograms from (B) were first sorted by decreasing frequency, then assembled on this plot for comparison. The slope of the curve is an indicator of the diversity (or lack thereof) in the corresponding distributions.

Bias in binding site selection due to size (Figure S2) or sequence (Figure 3D) can potentially be amplified by solver-specific tendencies: when faced with solutions of equal score and no additional constraints, the solver generates outcomes in an order that reflects its intrinsic algorithms, more akin to local exploration than direct sampling. This implies that the distribution of the binding sites involved in the first few solutions, generated among a large set of optimally scored solutions (for instance, those containing 6 binding sites), may not accurately represent the distribution of the entire optimal set. To create a sample of the best solutions with a more balanced distribution of binding sites, one could ask the solver for a much larger number of solutions and filter this set to homogenize the representation of binding sites. However, this is computationally wasteful, as many of the generated solutions are discarded.

Generation of more representative solution sets

We address the issue of sampling bias by making a minor adjustment to our integer linear programming model. By maintaining a count of the binding sites included in previously generated solutions, we adjust the objective of the integer linear program to favor those binding sites that are underrepresented compared to the average. Concretely, we modify the objective to $\sum_{i\in[0,\,n],\,j\in[1,\,n]} c_{ij}\ x_{ij}$ where c_{ij} is set to 1 for j corresponding to binding sites more represented than the average so far, and to $1+\epsilon$ for binding sites less represented than the average. The value of ϵ is irrelevant if it is sufficiently small to never allow the score of a solution with fewer binding sites to overcome the score of a solution with more binding sites (any positive value that is smaller than $1/n$ meets these criteria). We apply this change after every generated solution. This change allows the solver to break ties between solutions in a different way each time. We refer to the original solution method as "solver order," because the specific solver implementation determines how ties are broken. We refer to our modified version of this approach that directs the algorithm towards the use of underrepresented sites as "diversity-driven order," because the addition of the weighting terms controls how the solver breaks ties when selecting which binding site to include (Figure 4A).

Whereas the solver order strategy leads to a pool of solutions that never represents every binding site equally (Figure 4B), we found that the diversity-driven strategy is effective at generating solutions in an order that transiently improves inclusivity in binding site representation (Figure 4C). In this example, $|R|$ = 10 so equal representation of binding sites would have their frequency of appearance at 0.1. Both approaches ultimately result in the same binding site distribution because in the limit where they generate the maximum number of top-scoring optimal solutions, the two strategies produce the same set of solutions. Critically, what is different is the order in which these solutions are selected before this maximum is reached. To summarize the results across the different solution methods, we calculated the entropy of the solver order and diversity-driven order strategies (Figure 4D). Higher entropy values correspond to more diverse solutions, where the theoretical maximum is $\log(|R|)$, reached only if all binding sites have been uniformly involved (with frequency $1/|R|$) in the solutions. The diversity-driven order solutions exhibited a rapid increase in entropy that was maintained across many rounds of binding site selection (Figure 4D), demonstrating its efficacy in generating solutions with broad representation. The diversity-driven order also produced higher entropy distributions than the solver order for libraries with size differences among binding sites (Figure S3).

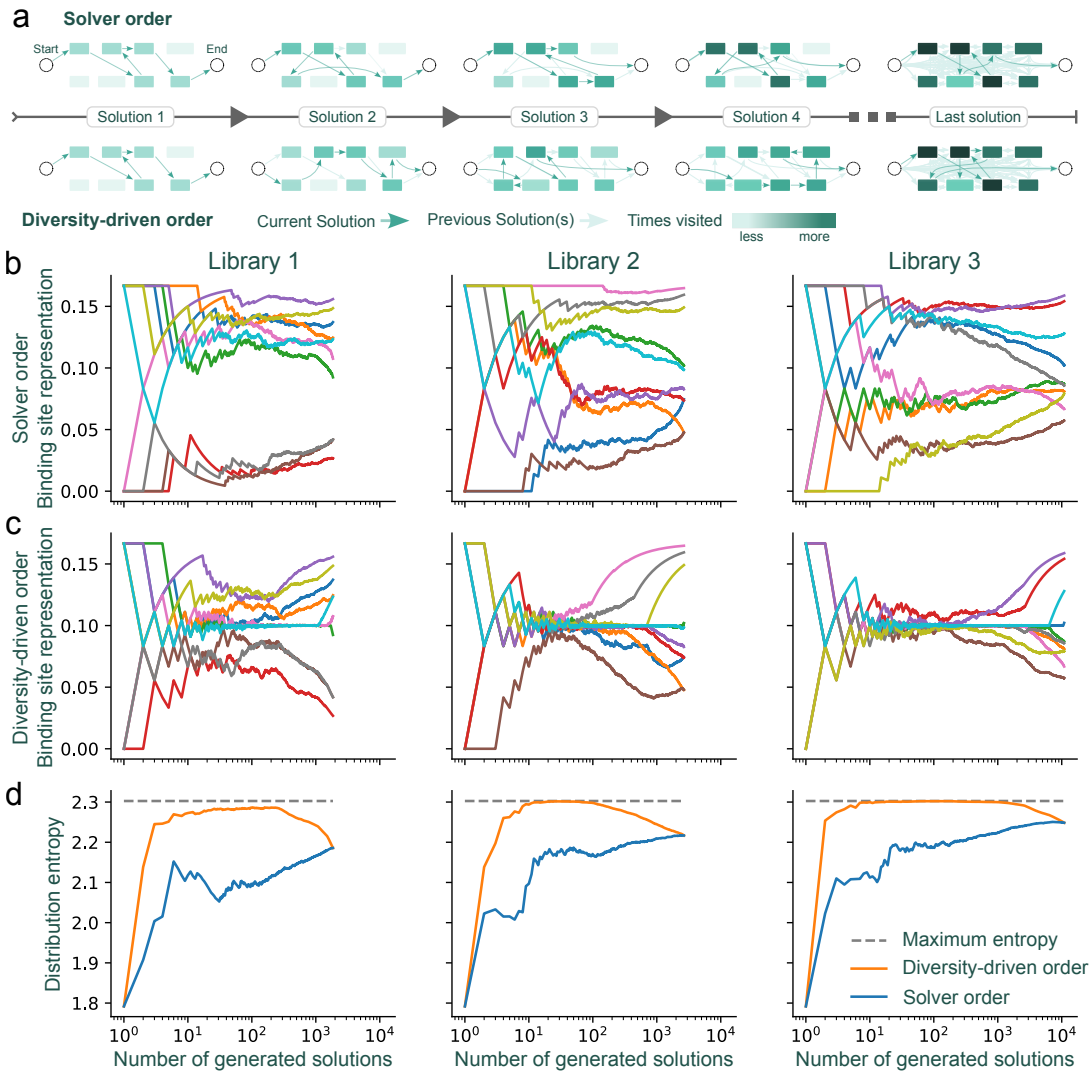

Figure 4. A minor modification to the model fosters a diverse representation of binding sites as solutions are returned. (A) Schematic representing binding site selection frequencies using solver order (top) or diversity-driven order (bottom). The color intensity of each vertex represents the absolute counts with which it was included in a solution. (B) Solver order and (C) diversity-driven order approach generates different frequencies of binding site representation. Each color represents a different binding site ($|R|$ = 10 binding sites). The three columns represent the three libraries presented in Fig. 3, with respective number of top-scoring solutions of 1854, 2654, and 10,922. (D) The distribution entropy of the solver order and diversity-driven order strategies. In all cases, the entropy of the diversity-driven order is the same or higher than the entropy of the solver order.

Incorporating positional constraints for promoter design

We next incorporated more constraints into the solver to facilitate the inclusion of sequence elements at fixed positions. As an example, we focused on the design of bacterial promoters. Sigma factor recognition sites are crucial elements in bacterial promoters, and they typically appear around -35 and -10 base pairs away from the transcriptional start site (52). Incorporating constraint at specific locations forces the solver to place user-defined sites within dense arrangements of binding sites on the forward strand (Figure 5A-5B, see Methods for the implementation of the constraints). We designed the constraints to be flexible enough that the user can specify arbitrary ranges, for example specifying that the downstream element should start between the –9 and –11 positions relative to the end of the sequence. This option for incorporating positional flexibility helps the solver to find feasible solutions that align with user-defined

constraints and allows for the specification of multiple fixed sequence elements (or pairs thereof) to be positioned at defined proximal distances from one another. For example, using this strategy, the solver can design dense arrays that contain tightly interlocked primary and alternative sigma factor recognition elements (Figure 5C-D). In nature, promoters often overlap, resulting in transcription start sites that are shared or close to each other, each linked to specific sigma factor recognition sites upstream (1). The positioning and spacing of these sites is shaped by the biophysical constraints of the corresponding sigma factor (53). Incorporating multiple sigma factor pairings into synthetic promoters helps sustain transcriptional strength across diverse environments and transient stress responses, which can be valuable for biotechnological applications (26,54). As an example, we incorporated three distinct promoters within a single dense array, featuring consensus sequences for σ^D, σ^S, and σ^N. These sigma factors are linked with housekeeping genes, stationary phase and the general stress response, and nitrogen metabolism, respectively. This combination enables the downstream genes of this synthetic promoter to theoretically respond to variations in any of these conditions. With fixed sequence lengths and positional constraints, we used random libraries of binding sites to generate 350 dense arrays. The solver successfully returned solutions that placed specified binding sites in predetermined positions, demonstrating the approach's feasibility on a library scale (Figure 5D).

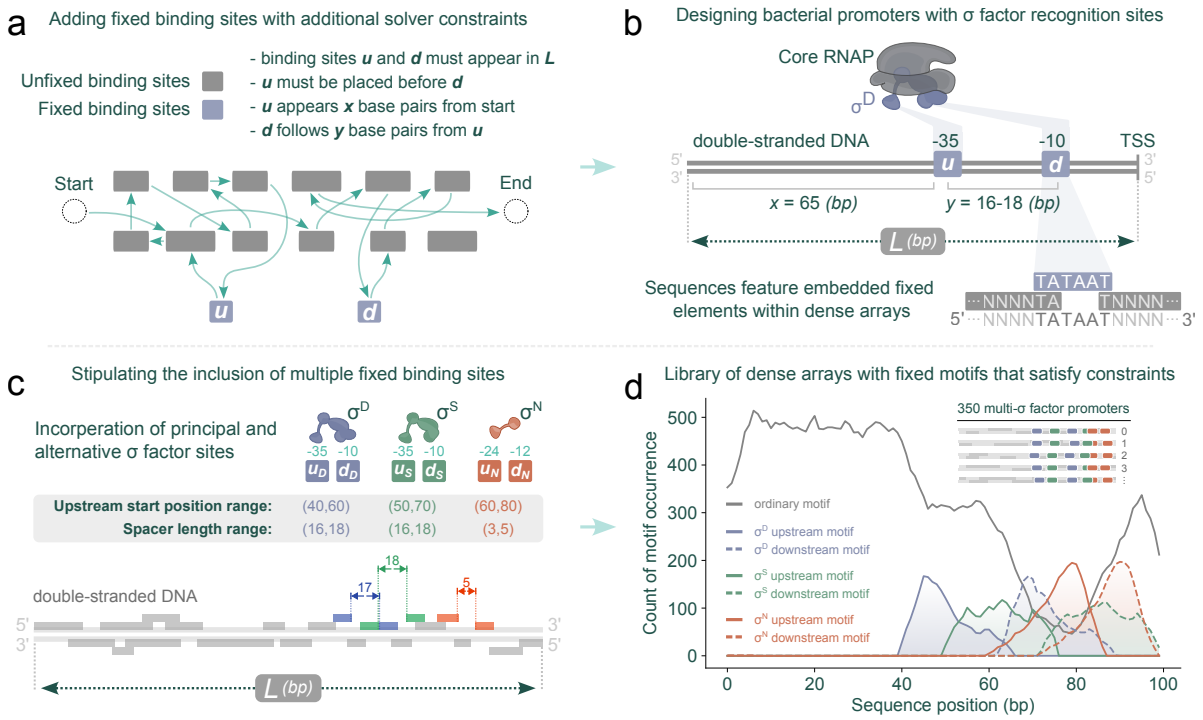

Figure 5. Stipulating positional constraints to design bacterial promoters. (A) By applying additional constraints, the solver is guided to create sequences with user-defined binding sites at predetermined locations. u, upstream site; d, downstream site. (B) This technique enables interweaving of dense binding site regions around -35 and -10 sigma factor recognition sites. (C) Extending this, the solver can incorporate both primary and alternative sigma factor binding sites, placed within densely packed binding sites. (D) Summary of results from 350 random binding site libraries. Each library was used to generate an output sequence of $L = 100$ base pairs. Each library contained $|R| = 20$ randomly generated binding sites, with sizes uniformly chosen between 5 to 15 base pairs. Alongside the ordinary binding sites in each library, six additional fixed binding sites were introduced with varying constraints: TTGACA/TATAAT, with its upstream position within 40-60 base pairs from the beginning of the sequence and a spacer length of 16-18 base pairs; TTGACA/TATACT, with its upstream position within 50-70 base pairs and also with a spacer length of 16-18 base pairs; and TGGCAGG/TTGCA, with its upstream position within 60-80 base pairs, but with a shorter spacer length of 3-5 base pairs. The total number of random libraries generated was 700: in half of the cases, the solver reported that no solution satisfying the constraints existed.

Transcription factor binding sites play varied roles in gene expression; for instance, some activate while others repress, depending on their associated regulator. Notably, binding sites for activators typically lie upstream of the -35 sigma factor recognition element, with their effectiveness diminishing if they are placed further downstream (23,52). To demonstrate how the model can accommodate this, we classified random binding sites as either activators or repressors, and then adjusted the model to position activators more upstream and repressors more downstream (Figure 6A, see Methods for the implementation). Modifying the linear objective in this way provides an additional layer of optimization to the solver, where otherwise equally scored solutions (e.g., two dense arrays that each can fit 19 binding sites) will be differentiated based on where activators and repressors appear in their sequences (Figure 6B).

Figure 6. Modifying the linear integer programming model objective to include a position bias for activators and repressors. (A) Classifying binding sites as either activators or repressors and then modifying the model to favor activators or repressors by minimizing/maximizing the position where they appear in the sequence. (B) The histogram was generated from data obtained from 120 random libraries, each consisting of |R| = 20 binding sites. These binding sites, classified randomly either as activators or repressors, varied in length from 5 to 15 base pairs, and were used to assemble sequences of L = 100 base pairs. Additionally, two 6-base pair binding sites, representing sigma factor recognition elements, were included with specific constraints: the downstream element was positioned between positions 89 and 91 (corresponding to the -10 site), with a spacer of 16-18 base pairs (for the -35 site) between the two elements.

Discussion

Dense arrangements of DNA-protein binding sites within nucleotide sequences can collectively influence downstream transcription rates or initiate biomolecular interactions. Despite the prevalence of such sequences in nature, rapidly generating large stretches of DNA with overlapping binding sites remains a challenge. Here, we present a computational method to pack nucleotide sequence binding sites into dense arrays (Figure 1). We initially formulated the task as a problem that we denote as the nucleotide SPP, which we showed to be NP-hard. The SPP takes its place within the classical NP-hard problems, in that its relation with the Shortest Common Superstring mirrors the relation of other pairs of classic NP-hard problems, such as Maximum Coverage and Set Cover, Multiple Knapsack and Bin Packing, or Orienteering Problem and Travelling Salesman Problem. We then reduced the task to an OP with integer distances, which we expressed as a problem that can be solved using integer linear programming—a highly expressive framework that is amenable to the inclusion of user-defined constraints, while also enabling the use of modern and efficient solvers. Although the task is still NP-hard, this approach allowed us to dramatically increase the computational efficiency of obtaining solutions over brute force approaches. Importantly, for biologically realistic parameters (e.g., sequence lengths of L = 50-300 base pairs with several dozen binding sites), this formulation can be solved in seconds (Figure 2). In contrast, a greedy polynomial-time method

was only able to reach the optimal solution when the problem was at its easiest (e.g., with few binding sites in a long sequence involving little to no compression (Figure S1)).

By enumerating the best solutions in order, we found that there can be a multitude of sequences that achieve optimal packing for a given library and sequence size (Figure 3). However, there is a strong bias in the representation of binding sites in the set of optimal solutions, where shorter binding sites appear more frequently (Figure S2) and sequence-specific effects impact binding site representation (Figure 3). While we highlight here how sampling binding sites from a large set can produce a library of distinct sequence variants, the problem of sampling is independent of the SPP. Sampling efforts can be customized in several ways to generate sequences that meet specific binding site representation requirements. This can be achieved either by modifying the integer linear programming model itself, such as by assigning different weights to various binding sites in the solution score, or by iteratively adding extra constraints to manage the distribution of the resulting sequences, as demonstrated in our approach (Figure 4, Figure S3). We also demonstrate how the solver can incorporate specific sequence elements at fixed positions within the dense arrays through additional constraints. This method allows for the strategic placement of binding sites (or pairings thereof) at defined locations, enabling the scalable design of bacterial promoters with intricate binding site architectures (Figure 5). Moreover, solutions can be further refined to resemble natural promoters more closely by adding incentives based on the functional role of binding sites, such as distinguishing between activators and repressors. This approach guides their preferential placement either upstream or downstream within the dense array (Figure 6).

It is also worth noting that the presented method can extend beyond cis-regulatory regions and nucleotides. It can, in theory, densely pack strings made from any set of characters. Therefore, it is amenable to forming continuous stretches of virtually any primary biological sequence. The advantage of this graph-based approach is expected to increase as the alphabet size increases because our formulation is independent of the size of the alphabet, in contrast to methods modeling each letter individually. Beyond biological applications, the SPP introduced here could find use cases in data compression, as seen with its variant the SCS problem (55,56).

Further work could explore designing solver constraints that account for additional structural and proximal factors influencing gene expression. This includes exploring how a regulator's bound, helical positioning on the DNA influences its distance to RNA polymerase, thus affecting its regulatory effectiveness (52). Another consideration for bacterial promoter design is the prevalence of transcription factors that operate as homodimers, which form binding sites that resemble “spaced dyads,” characterized by less conserved middle regions (52). The dense arrangement of these dyads may present additional constraints and challenges to the solver, necessitating more nuanced and sophisticated design strategies. However, many binding sites are defined as fuzzy, in contrast to the strict motifs associated with restriction enzymes, thus many arrangements of densely packed binding sites would likely give rise to nuanced and complex dynamics in cis-regulatory regions.

The design and application of DNA sequences composed of discrete motif elements is reliant on the availability of accurate and biologically relevant binding site annotations. Fortunately, continued characterization of DNA-protein binding profiles across organisms, sourced from techniques like ChIP-seq, DAP-seq, ATAC-seq, and HT-SELEX, will support the curation of binding sites associated with proteins and cellular mechanisms. Equally applicable to large-scale libraries and more focused studies, the nucleotide string packing approach we introduce here serves as a resource for designing information-dense promoter sequences, which could then be used to map how intricate binding site configurations influence gene expression or biomolecular mechanisms.

Methods

Formulation of the SPP as an integer linear program

Given that the SPP is NP-hard, there is no known algorithm that can solve it in a time that is polynomial in the size of the input, making solving medium to large instances impractical with naïve methods. However, many NP-hard problems have practical significance, which has led to the development of optimized solvers (43), designed to manage the combinatorial explosion and return optimal solutions (as time and memory constraints permit). Although these solvers might not return solutions in large instances within their allocated time and memory limits, they can offer surprising efficiency for smaller problems. Many numerical solvers have been specifically developed to tackle the NP-hard integer linear programming problem, which restricts linear programming to integer variables. We formulated the SPP as an integer linear programming problem by reducing it to a variant of the Travelling Salesman Problem (TSP) known as the Orienteering Problem (OP) (40,44,45). The TSP, a classic example of an NP-hard problem, involves determining the shortest possible route that visits every vertex of a graph. In the OP, there is an allocated time budget to traverse the graph, and the goal is to visit as many distinct vertices as possible within that time budget. The OP's relation to the TSP mirrors the SPP's relation to the SCS problem. In fact, this relationship is shared by several other pairs of classical NP-hard problems, where one problem consists of minimizing the cost of covering all possible elements (e.g. Set Cover, Bin Packing, Traveling Salesman, Shortest Common Superstring), and the other consists of maximizing the number of elements covered within an arbitrary given cost (respectively: Maximum Coverage, Multiple Knapsack, Orienteering Problem, and the String Packing Problem which to our knowledge had not been named yet). Note that although similar, these pairs of problems are not equivalent nor can they be solved or approximated with the same efficiency. For instance, while Maximum Coverage admits a constant-factor polynomial-time approximation algorithm (58), it has been demonstrated that Set Cover cannot be approximated to $\left(1 - o(1)\right) \ln n$ unless P = NP (59). We can map any SPP to an OP by representing binding sites as vertices on a complete directed graph.

We define a metric d_{ij} to represent the cost associated with going from one nucleotide string to another in the graph. Specifically, the edge that connects string a and string b is given a natural integer weight, which represents the offset needed for the prefix of string b to match the suffix of string a (Figure 1B). In other words, it describes the minimal number of spaces that string b needs to be shifted to the right to coincide and cover the end of string a. For example, the directed edge from AGC to CAG is weighted 2, while the reverse edge is weighted 1. If no suffix of a matches a prefix of b, then we consider that the empty suffix of a matches the empty prefix of b, such that CGT to CAG is weighted 3. Importantly, a suffix of a needs to match a prefix of b, such that b overhangs at the end of a (possibly by 0 characters), but cannot stop before the end of a, for example, the directed edge from AGCAG to GC must be weighted 4, not 1 (1 shift does not allow GC to overhang in the end of AGCAG). This requirement allows the path to resume from the last vertex visited (here, GC) without any additional constraints. It also makes the metric d_{ij} satisfy the triangle inequality. Finally, we create two additional vertices: a start vertex with distance 0 to all other vertices, and an end vertex, such that the distance of any binding site to the end vertex is the length of this binding site (Figure 1C). Maximizing the number of binding sites included within a string of length L is then equivalent to maximizing the number of graph vertices visited between the start and end vertices, with a sum of edge weights not larger than the total budgeted length L. This corresponds to solving the OP on this graph. Because all the distances are integers, this problem can be modeled as an integer linear program. We adapt the Miller – Tucker – Zemlin TSP formulation (60) to consider the specificities of our problem, as follows:

Variables

We represent a path in the graph with binary variables. For every edge between two binding sites, we create

one binary variable to indicate if this edge is taken, that is, if the second binding site comes directly after the first one in the final sequence (with the specified overlap). This creates n^2+n+1 binary variables: with the index 0 representing both the start and end vertex, we have $n(n-1)$ edges between internal vertices (x_{ij} with $i \neq j$), n edges between the start and all internal vertices (x_{0j}), n edges between all internal vertices and the end vertex (x_{i0}), as well as one edge between the start and the end (x_{00}) (taken if and only if no binding sites can fit in the solution, that is, if the sequence length L is shorter than all binding sites).

Objective

The solver's task is to maximize the number of vertices visited by the path in the graph. This is equivalent to maximizing the sum of binary variables representing edges that enter an internal vertex (not the end vertex): $\sum_{i\in[0,\,n],\,j\in[1,\,n]} x_{ij}$. The following constraints impose that $x_{ij} = 1$ if and only if the path contains the edge going from i to j.

Constraints

1. The path must start at the start vertex and end at the end vertex: $\sum_{i\in[1,\,n]} x_{i0} = 1$ and $\sum_{j\in[1,\,n]} x_{0j} = 1$.
2. Vertices cannot be visited more than once: $\forall i \in [1,\, n], \sum_{j\in[1,\,n]} x_{ij} \leq 1$ and $\forall j \in [1,\, n], \sum_{i\in[1,\,n]} x_{ij} \leq 1$.
3. Any given internal vertex is entered and exited the same number of times: $\forall k \in [1,\, n], \sum_{i\in[1,\,n]} x_{ik} = \sum_{j\in[1,\,n]} x_{kj}$.
4. The total length of the path is not longer than the desired final sequence: $\sum_{i\in[1,n],\,j\in[0,n]} d_{ij}\ x_{ij} \leq L$ where d_{ij} represents the metric described above.
5. Miller – Tucker – Zemlin subtour elimination constraints introduce one additional integer variable per binding site, $\forall i \in [1,\, n]$, $u_i \in [1,\, n]$, and $n(n-1)$ additional constraints: $\forall i \in [1,\, n]$, $\forall j \in [1,\, n]$, $u_i - u_j + 1 \leq n(1 - x_{ij})$. These additional variables and constraints implement a counter that must increase along the path, which prevents subtours (loops disconnected from the main path).

Reconstructing the solution from the x_{ij} variables yields solutions such as those pictured in Figure 1D. The expressivity of integer linear programming allows us to modify this base model to include additional constraints and specifications. For example, double-stranded optimization (allowing binding sites to appear not only in one strand but also as their reverse-complement on the other) can be implemented by duplicating every vertex, reverse-complementing the sequences, and modifying constraint 2 to prevent the algorithm from visiting both a binding site and its reverse complement. Note that while the single-strand version of the SPP mirrors the OP, the double-strand version of the problem is not strictly an OP, as it involves additional constraints such as restricting the inclusion of both a binding site and its reverse complement. However, both single and double stranded versions can be solved efficiently using integer linear programming. We implemented the SPP integer linear programming model with the OR-Tools optimization toolbox (43), which is able to call open-source (CBC (61) or SCIP (62)) or commercial (Gurobi (63)) integer linear programming solvers.

<u>Implementation of the promoter constraints</u>

Promoter constraints are formulated in terms of absolute positions in the DNA sequence: these positions must then be encoded in the linear integer model. We achieve this with a set of n integer variables with values from 0 to L-1 (both included), and the following linear constraints:

$\forall i \in [0,\, n]$, $\forall j \in [1,\, n]$, $i \neq j$, $d_{ij}\, x_{ij} - (L-1)(1 - x_{ij}) \leq position_j - position_i \leq d_{ij}\, x_{ij} + (L-1)(1 - x_{ij})$ where $position_0 = 0$ (the position of the empty start vertex).

This follows a similar approach as the subtour elimination constraints, forcing $position_j - position_i$ to be equal to d_{ij} if and only if $x_{ij} = 1$ (otherwise, any value is possible).

With these position variables defined, we implement the promoter position constraints as follows:

- If binding site k corresponds to a fixed binding site, it must appear in the sequence: $\sum_{i \in [0,\, n],\, i \neq k} x_{ik} \geq 1$
- If binding site k corresponds to a fixed binding site that needs to start between positions α and β, then $\alpha \leq position_k \leq \beta$
- If binding sites k and l are the upstream and downstream elements of a promoter where the spacer length is constrained to be between α and β base pairs: $\alpha \leq position_l - length_l - position_k \leq \beta$ with $length_l$ being the length of binding site l

Implementation of the side bias

The side biases (making binding sites with upstream preference appear preferably upstream, respectively downstream) can be implemented using the position variables described in the previous subsection. We add the term $\pm position_i / K$ to the score function if binding site i has an upstream preference $(-)$ or a downstream preference $(+)$. These terms are added to an integer-step scoring function (the number of binding sites in the sequence). The value of the strictly positive constant K is thus irrelevant as long as the sum of all these terms cannot take a value outside of $[-0.5,\ 0.5]$: this ensures that two solutions with different numbers of binding sites cannot be reordered.

Source Code

The solver is implemented in an open-source library available at: https://gitlab.com/dunloplab/dense-arrays. The code used to generate the data and plot the figures of this paper is available on the "paper" branch of this repository.

Acknowledgements

We would like to thank Hang Zhou and Rayan Chikhi for helpful comments on the manuscript. This work was supported by the National Science Foundation (MCB-2143289 and MCB-2324909 to MJD). EJS received support from the National Institutes of Health under award number T32GM130546.

References

1. Mejía-Almonte C, Busby SJW, Wade JT, van Helden J, Arkin AP, Stormo GD, et al. Redefining fundamental concepts of transcription initiation in bacteria. Nat Rev Genet. 2020 Nov;21(11):699–714.
2. Sharon E, Kalma Y, Sharp A, Raveh-Sadka T, Levo M, Zeevi D, et al. Inferring gene regulatory logic from high-throughput measurements of thousands of systematically designed promoters. Nat Biotechnol. 2012 Jun;30(6):521–30.
3. LaFleur TL, Hossain A, Salis HM. Automated model-predictive design of synthetic promoters to control transcriptional profiles in bacteria. Nat Commun. 2022 Sep 2;13(1):5159.
4. Jensen D, Galburt EA. The Context-Dependent Influence of Promoter Sequence Motifs on Transcription Initiation Kinetics and Regulation. J Bacteriol. 2021 Mar 23;203(8):e00512-20.
5. Dijk D van, Sharon E, Lotan-Pompan M, Weinberger A, Segal E, Carey LB. Large-scale mapping of gene regulatory logic reveals context-dependent repression by transcriptional activators. Genome Res. 2017 Jan 1;27(1):87–94.
6. Rogers JK, Guzman CD, Taylor ND, Raman S, Anderson K, Church GM. Synthetic biosensors for precise gene control and real-time monitoring of metabolites. Nucleic Acids Res. 2015;43(15):7648–60.
7. Hossain GS, Saini M, Miyake R, Ling H, Chang MW. Genetic Biosensor Design for Natural Product Biosynthesis in Microorganisms. Trends Biotechnol. 2020;38:797–810.
8. Lupo O, Kumar DK, Livne R, Chappleboim M, Levy I, Barkai N. The architecture of binding cooperativity between densely bound transcription factors. Cell Syst [Internet]. 2023 Jul 31 [cited 2023 Aug 27]; Available from: https://www.sciencedirect.com/science/article/pii/S2405471223001850
9. Brodsky S, Jana T, Mittelman K, Chapal M, Kumar DK, Carmi M, et al. Intrinsically Disordered Regions Direct Transcription Factor In Vivo Binding Specificity. Mol Cell. 2020 Aug 6;79(3):459-471.e4.
10. Struhl K. Molecular mechanisms of transcriptional regulation in yeast. Annu Rev Biochem. 1989;58:1051–77.
11. Kazemian M, Pham H, Wolfe SA, Brodsky MH, Sinha S. Widespread evidence of cooperative DNA binding by transcription factors in Drosophila development. Nucleic Acids Res. 2013 Sep;41(17):8237–52.
12. Ishihama A. Prokaryotic genome regulation: multifactor promoters, multitarget regulators and hierarchic networks. FEMS Microbiol Rev. 2010 Sep 1;34(5):628–45.
13. Rydenfelt M, Garcia HG, Iii RSC, Phillips R. The Influence of Promoter Architectures and Regulatory Motifs on Gene Expression in Escherichia coli. PLOS ONE. 2014 Dec 30;9(12):e114347.
14. Gertz J, Cohen BA. Environment-specific combinatorial cis-regulation in synthetic promoters. Mol Syst Biol. 2009 Jan;5(1):244.
15. Yu TC, Liu WL, Brinck MS, Davis JE, Shek J, Bower G, et al. Multiplexed characterization of rationally designed promoter architectures deconstructs combinatorial logic for IPTG-inducible systems. Nat Commun. 2021 Jan 12;12(1):325.
16. Urtecho G, Insigne KD, Tripp AD, Brinck M, Lubock NB, Kim H, et al. Genome-wide Functional Characterization of Escherichia coli Promoters and Regulatory Elements Responsible for their Function [Internet]. bioRxiv; 2020 [cited 2023 Jun 15]. p. 2020.01.04.894907. Available from: https://www.biorxiv.org/content/10.1101/2020.01.04.894907v1
17. Tewhey R, Kotliar D, Park DS, Liu B, Winnicki S, Reilly SK, et al. Direct Identification of Hundreds of Expression-Modulating Variants using a Multiplexed Reporter Assay. Cell. 2016 Jun 2;165(6):1519–29.
18. Johns NI, Gomes ALC, Yim SS, Yang A, Blazejewski T, Smillie CS, et al. Metagenomic mining of regulatory elements enables programmable species-selective gene expression. Nat Methods. 2018 May;15(5):323–9.
19. Cuperus JT, Groves B, Kuchina A, Rosenberg AB, Jojic N, Fields S, et al. Deep learning of the regulatory grammar of yeast 5′ untranslated regions from 500,000 random sequences. Genome Res. 2017 Dec 1;27(12):2015–24.
20. Van Brempt M, Clauwaert J, Mey F, Stock M, Maertens J, Waegeman W, et al. Predictive design of sigma factor-specific promoters. Nat Commun. 2020 Nov 16;11(1):5822.
21. Cazier AP, Blazeck J. Advances in promoter engineering: Novel applications and predefined transcriptional control. Biotechnol J. 2021;16(10):2100239.
22. Blazeck J, Alper HS. Promoter engineering: Recent advances in controlling transcription at the most fundamental level. Biotechnol J. 2013;8(1):46–58.
23. Cox III RS, Surette MG, Elowitz MB. Programming gene expression with combinatorial promoters. Mol Syst Biol. 2007 Jan;3(1):145.
24. Inukai S, Kock KH, Bulyk ML. Transcription factor–DNA binding: beyond binding site motifs. Curr Opin Genet Dev. 2017 Apr 1;43:110–9.

25. Mogno I, Kwasnieski JC, Cohen BA. Massively parallel synthetic promoter assays reveal the in vivo effects of binding site variants. Genome Res. 2013 Nov 1;23(11):1908–15.
26. Wang Y, Liu Q, Weng H, Shi Y, Chen J, Du G, et al. Construction of Synthetic Promoters by Assembling the Sigma Factor Binding −35 and −10 Boxes. Biotechnol J. 2019;14(1):1800298.
27. Stewart AJ, Hannenhalli S, Plotkin JB. Why Transcription Factor Binding Sites Are Ten Nucleotides Long. Genetics. 2012 Nov 1;192(3):973–85.
28. Guazzaroni ME, Silva-Rocha R. Expanding the Logic of Bacterial Promoters Using Engineered Overlapping Operators for Global Regulators. ACS Synth Biol. 2014 Sep 19;3(9):666–75.
29. Amores GR, Guazzaroni ME, Silva-Rocha R. Engineering Synthetic cis-Regulatory Elements for Simultaneous Recognition of Three Transcriptional Factors in Bacteria. ACS Synth Biol. 2015 Dec 18;4(12):1287–94.
30. Monteiro LMO, Sanches-Medeiros A, Westmann CA, Silva-Rocha R. Modulating Fis and IHF binding specificity, crosstalk and regulatory logic through the engineering of complex promoters [Internet]. bioRxiv; 2019 [cited 2022 Dec 19]. p. 614396. Available from: https://www.biorxiv.org/content/10.1101/614396v1
31. Kotopka BJ, Smolke CD. Model-driven generation of artificial yeast promoters. Nat Commun. 2020 Apr 30;11(1):2113.
32. Sample PJ, Wang B, Reid DW, Presnyak V, McFadyen IJ, Morris DR, et al. Human 5′ UTR design and variant effect prediction from a massively parallel translation assay. Nat Biotechnol. 2019 Jul;37(7):803–9.
33. Taskiran II, Spanier KI, Christiaens V, Mauduit D, Aerts S. Cell type directed design of synthetic enhancers [Internet]. bioRxiv; 2022 [cited 2023 May 3]. p. 2022.07.26.501466. Available from: https://www.biorxiv.org/content/10.1101/2022.07.26.501466v1
34. Seo E, Choi YN, Shin YR, Kim D, Lee JW. Design of synthetic promoters for cyanobacteria with generative deep-learning model. Nucleic Acids Res. 2023 May 29;gkad451.
35. Gosai SJ, Castro RI, Fuentes N, Butts JC, Kales S, Noche RR, et al. Machine-guided design of synthetic cell type-specific cis-regulatory elements [Internet]. bioRxiv; 2023 [cited 2023 Aug 13]. p. 2023.08.08.552077. Available from: seqprop
36. Nikolados EM, Oyarzún DA. Deep learning for optimization of protein expression. Curr Opin Biotechnol. 2023 Jun 1;81:102941.
37. Sanabria M, Hirsch J, Poetsch AR. Distinguishing word identity and sequence context in DNA language models [Internet]. bioRxiv; 2023 [cited 2023 Jul 31]. p. 2023.07.11.548593. Available from: https://www.biorxiv.org/content/10.1101/2023.07.11.548593v1
38. Zhang P, Wang H, Xu H, Wei L, Liu L, Hu Z, et al. Deep flanking sequence engineering for efficient promoter design using DeepSEED. Nat Commun. 2023 Oct 9;14(1):6309.
39. Garey MR, Johnson DS. Computers and Intractability: a guide to the theory of NP-Completeness. New York: Freeman; 1985. (Series of books un the mathematical sciences).
40. Gunawan A, Lau HC, Vansteenwegen P. Orienteering Problem: A survey of recent variants, solution approaches and applications. Eur J Oper Res. 2016 Dec 1;255(2):315–32.
41. Maier D, Storer JA. A Note on the Complexity of the Superstring Problem. Computer Science Laboratory: Princeton University; 1977 Oct. Report No.: 233.
42. OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. 2023. Entry A000522. Available from: https://oeis.org/A000522
43. Google. OR-Tools, v9.7. 2023; Available from: https://developers.google.com/optimization/
44. Golden BL, Levy L, Vohra R. The orienteering problem. Nav Res Logist NRL. 1987;34(3):307–18.
45. Tsiligirides T. Heuristic Methods Applied to Orienteering. J Oper Res Soc. 1984 Sep 1;35(9):797–809.
46. Tierrafría VH, Rioualen C, Salgado H, Lara P, Gama-Castro S, Lally P, et al. RegulonDB 11.0: Comprehensive high-throughput datasets on transcriptional regulation in Escherichia coli K-12. Microb Genomics. 2022;8(5):000833.
47. Christofides N. Worst-case analysis of a new heuristic for the travelling salesman problem. Graduate School of Industrial Application, Carnegie-Mellon University; 1976 Feb. Report No.: 388.
48. van Bevern R, Slugina VA. A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem. Hist Math. 2020 Nov 1;53:118–27.
49. Сердюков АИ. О некоторых экстремальных обходах в графах. Дискретный Анализ И Исследование Операций. 1978;(17):76–9.
50. Turner JS. Approximation algorithms for the shortest common superstring problem. Inf Comput. 1989 Oct 1;83(1):1–20.
51. Englert M, Matsakis N, Veselý P. Improved approximation guarantees for shortest superstrings using cycle classification by overlap to length ratios. In: Proceedings of the 54th Annual ACM SIGACT Symposium on

Theory of Computing [Internet]. New York, NY, USA: Association for Computing Machinery; 2022 [cited 2023 Oct 12]. p. 317–30. (STOC 2022). Available from: https://doi.org/10.1145/3519935.3520001
52. van Hijum SAFT, Medema MH, Kuipers OP. Mechanisms and Evolution of Control Logic in Prokaryotic Transcriptional Regulation. Microbiol Mol Biol Rev MMBR. 2009 Sep;73(3):481–509.
53. Bervoets I, Van Brempt M, Van Nerom K, Van Hove B, Maertens J, De Mey M, et al. A sigma factor toolbox for orthogonal gene expression in Escherichia coli. Nucleic Acids Res. 2018 Feb 28;46(4):2133–44.
54. Wang Y, Shi Y, Hu L, Du G, Chen J, Kang Z. Engineering strong and stress-responsive promoters in Bacillus subtilis by interlocking sigma factor binding motifs. Synth Syst Biotechnol. 2019 Dec 1;4(4):197–203.
55. Ilie L, Popescu C. The Shortest Common Superstring Problem and Viral Genome Compression. Fundam Informaticae. 2006;73(1–2):153–64.
56. Storer JA. Data compression: methods and theory. Computer Science Press, Inc.; 1987.
57. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by Simulated Annealing. Science. 1983 May 13;220(4598):671–80.
58. Hochbaum D, editor. Approximation Algorithms for NP-Hard Problems. 1st edition. Boston: Course Technology; 1996. 624 p.
59. Dinur I, Steurer D. Analytical approach to parallel repetition. In: Proceedings of the forty-sixth annual ACM symposium on Theory of computing [Internet]. New York, NY, USA: Association for Computing Machinery; 2014 [cited 2024 Jan 24]. p. 624–33. (STOC '14). Available from: https://doi.org/10.1145/2591796.2591884
60. Miller CE, Tucker AW, Zemlin RA. Integer Programming Formulation of Traveling Salesman Problems. J ACM. 1960 Oct 1;7(4):326–9.
61. Forrest J, Ralphs T, Santos HG, Vigerske S, Forrest J, Hafer L, et al. coin-or/Cbc: Release releases/2.10.10 [Internet]. Zenodo; 2023 [cited 2023 Oct 3]. Available from: https://zenodo.org/record/7843975
62. Bestuzheva K, Chmiela A, Müller B, Serrano F, Vigerske S, Wegscheider F. Global Optimization of Mixed-Integer Nonlinear Programs with SCIP 8. 2023 [cited 2023 Oct 3]; Available from: https://arxiv.org/abs/2301.00587
63. Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. [Internet]. 2023. Available from: https://www.gurobi.com