Generating information-dense nucleotide 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

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. 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 can help interrogate how complex binding site arrangements impact either gene expression or biomolecular mechanisms in varied cellular contexts.

Keywords

string packing, transcription factor binding sites, integer linear programming, Orienteering Problem, DNA-protein binding interactions, molecular sequence design

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 fewer binding sites are thought to permit higher overall transcription levels even when transcription factor abundance is low [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 some 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), drawing parallels with 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 fits the three elements AA, GG, AGA (among others), due to the non-contiguous nature of subsequences.

After demonstrating 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 (e.g., ensuring each binding site from a set appears in the greater sequence just once), which can be used to tailor the specifics of the output nucleotide sequence. We also 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. Equally applicable to large-scale libraries and more focused studies, this computational approach serves as a valuable resource for designing unique sequences 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 $w = w_0 x w_1$ (Fig. 1a).

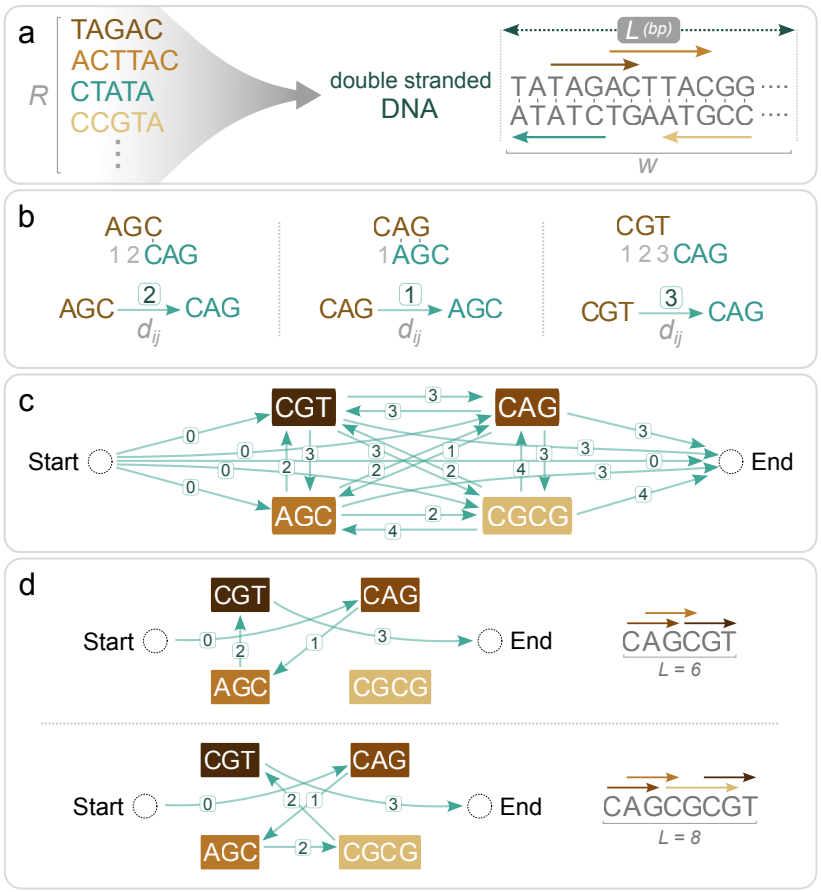

Figure 1. Formulation of the nucleotide String Packing Problem (SPP) as an Orienteering Problem (OP). **(a)** The SPP consists of fitting as many different strings from a library as possible inside a string of a given length L. base pairs, bp. **(b)** We define an asymmetrical metric between strings 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 less 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.

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 [44], 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) [40,45]. 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. We can map any SPP to an OP by representing binding sites as vertices on a complete and 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* (Fig. 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 (Fig. 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 [46] 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 Fig. 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 [44], which is able to call open-source (CBC [47] or SCIP [48]) or commercial (Gurobi [49]) integer linear programming solvers.

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. An alternative brute force approach to solve the equivalent OP formulation would be to enumerate all possible paths in the graph visiting each vertex at most once (Fig. 1c): there are about 10^{65} such paths (as calculated using the approach in Ref. [50]) which is still astronomically large, but much less so. Thus, the same problem can be formulated in different ways, and the amount of computational work needed to solve them can be very different. Due to the NP-hardness of the problem, the algorithms in the integer linear programming solvers 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 this size 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.

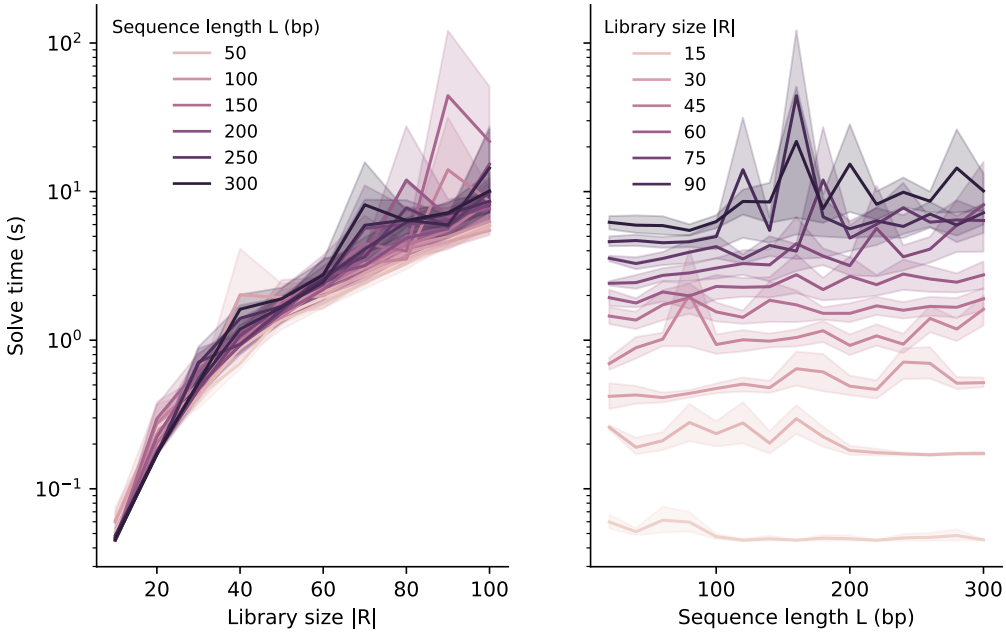

Figure 2. Solve time as a function of **(a)** library size, $|R|$, and **(b)** sequence length, L. All experiments were run 10

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 [51] 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 (Fig. 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 (Fig. 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 [52–54]. Approximation algorithms also exist for the SCC problem [42]. 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 [55]. 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. 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 (experimentally, 60% of the optimal in the worst cases), with large library sizes ($|R|$) and compact sequence lengths (L) posing challenges that only the integer linear programming formulation was able to tackle (Fig. 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.

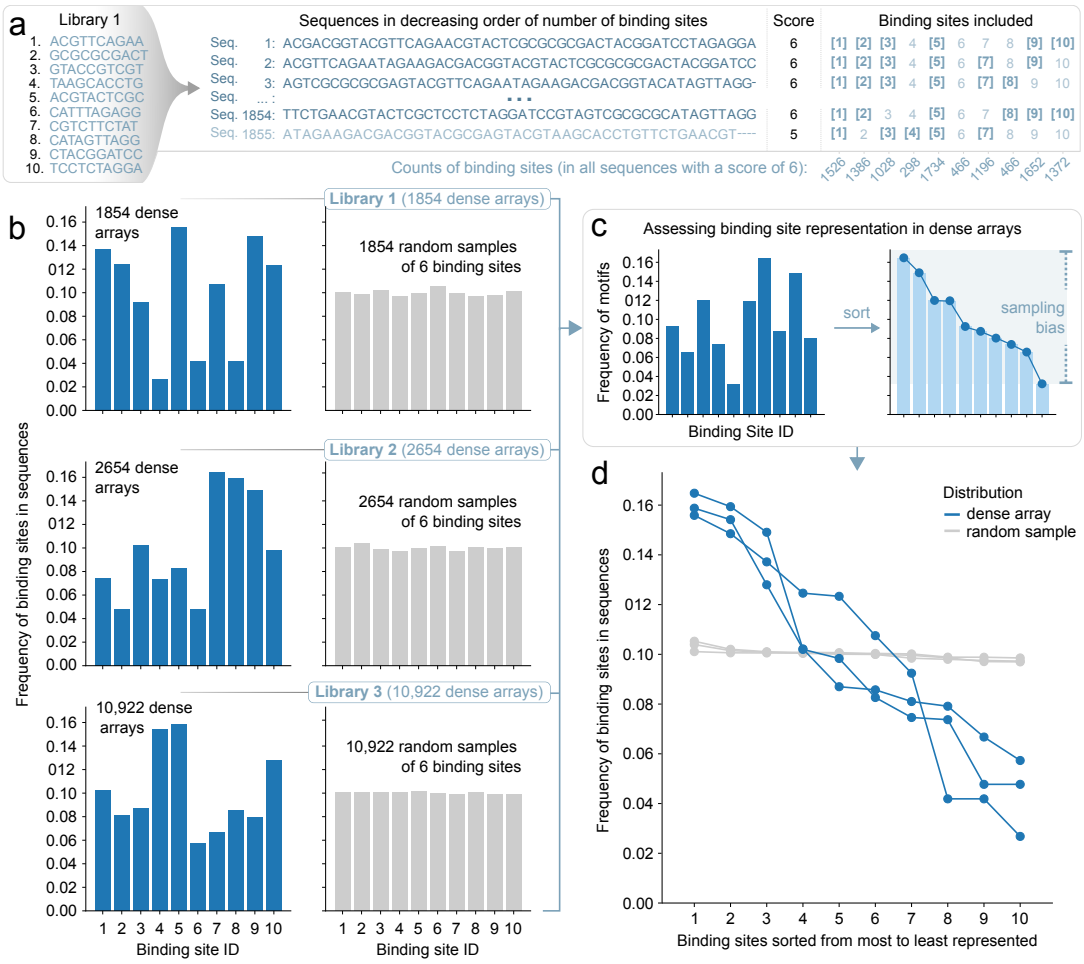

Figure 3. Assessing binding site representation across dense arrays. **(a)** For the $|R| = 10$ randomly generated binding sites of Library 1 we found 1854 optimal sequences (each fitting 6 binding sites within $L = 50$ bp). We then tallied up the binding sites involved in these sequences. **(b)** The 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.

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 (Fig. 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) (Fig. 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 (Fig. 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 (Fig. 3b-c). 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 (Fig. 3d).

Bias in binding site selection due to size (Fig. S2) or sequence (Fig. 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 first few solutions generated among a large set of solutions that have optimal scores (e.g., contain 6 binding sites) may not represent a fair sample of the entire optimal set, especially concerning the distribution of the individual binding sites involved. To create a sample of the best solutions with a more uniform distribution of binding sites, one could ask the solver for a much larger number of solutions than one needs, 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 precise 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 (Fig. 4a).

Whereas the solver order strategy leads to a pool of solutions that never represents every binding site equally (Fig. 4b), we found that the diversity-driven strategy is effective at generating solutions in an order that transiently improves inclusivity in binding site representation (Fig. 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 (Fig. 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 (Fig. 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 (Fig. S3).

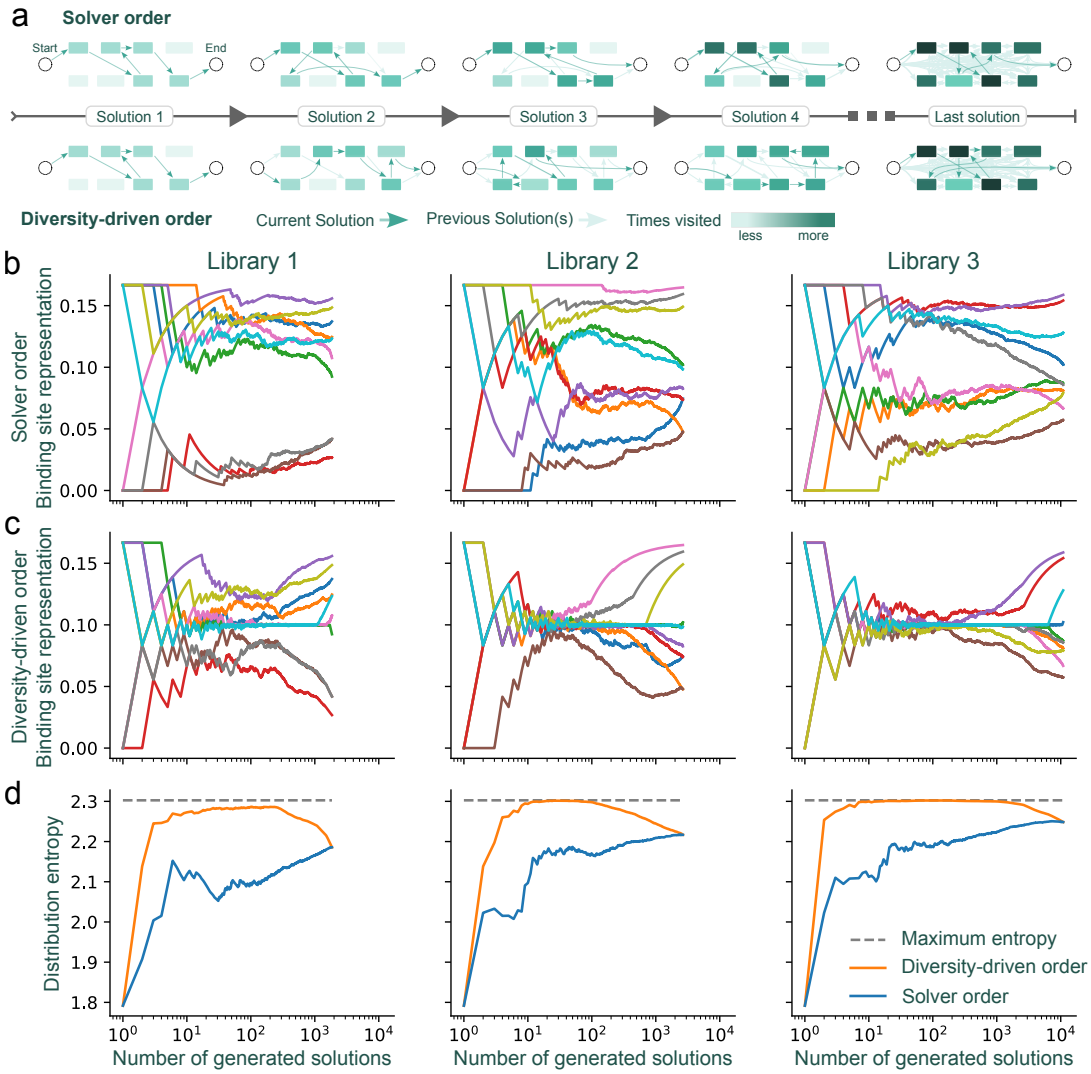

Figure 4. A minor modification to the model fosters a diverse representation of binding sites as solutions are returned. **(a)** Schematic representing binding site selection frequencies using solver order (top) or diversity-driven order (bottom). The color intensity of each vertex represents the absolute counts with which it was included in a solution. **(b)** Solver order and **(c)** diversity-driven order approach generate 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.

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 (Fig. 1). We initially formulated the task as a problem that we denote as the nucleotide SPP, which we showed to be NP-hard. 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 (Fig. 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 (Fig. 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 (Fig. 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 (Fig. S2) and sequence-specific effects impact binding site representation (Fig. 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 (Fig. 4, Fig. S3).

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 (e.g., dense assembly of peptide fragments into larger protein complexes). The advantage of this graph-based approach is expected to increase as the alphabet size increases because our formulation is independent on 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 [56,57].

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 sequences, which could then be used to map how intricate binding site configurations influence gene expression or biomolecular mechanisms.

Source Code

Code is available at: https://gitlab.com/dunloplab/dense-arrays

Acknowledgements

We would like to thank Hang Zhou for her helpful comments 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. *et al.* Redefining fundamental concepts of transcription initiation in bacteria. *Nat. Rev. Genet.* **21**, 699–714 (2020).
2. Sharon, E. *et al.* Inferring gene regulatory logic from high-throughput measurements of thousands of systematically designed promoters. *Nat. Biotechnol.* **30**, 521–530 (2012).
3. LaFleur, T. L., Hossain, A. & Salis, H. M. Automated model-predictive design of synthetic promoters to control transcriptional profiles in bacteria. *Nat. Commun.* **13**, 5159 (2022).
4. Jensen, D. & Galburt, E. A. The Context-Dependent Influence of Promoter Sequence Motifs on Transcription Initiation Kinetics and Regulation. *J. Bacteriol.* **203**, e00512-20 (2021).
5. Dijk, D. van *et al.* Large-scale mapping of gene regulatory logic reveals context-dependent repression by transcriptional activators. *Genome Res.* **27**, 87–94 (2017).
6. Rogers, J. K. *et al.* Synthetic biosensors for precise gene control and real-time monitoring of metabolites. *Nucleic Acids Res.* **43**, 7648–7660 (2015).
7. Hossain, G. S., Saini, M., Miyake, R., Ling, H. & Chang, M. W. Genetic Biosensor Design for Natural Product Biosynthesis in Microorganisms. *Trends Biotechnol.* **38**, 797–810 (2020).
8. Lupo, O. *et al.* The architecture of binding cooperativity between densely bound transcription factors. *Cell Syst.* (2023) doi:10.1016/j.cels.2023.06.010.
9. Brodsky, S. *et al.* Intrinsically Disordered Regions Direct Transcription Factor In Vivo Binding Specificity. *Mol. Cell* **79**, 459-471.e4 (2020).
10. Struhl, K. Molecular mechanisms of transcriptional regulation in yeast. *Annu. Rev. Biochem.* **58**, 1051–1077 (1989).
11. Kazemian, M., Pham, H., Wolfe, S. A., Brodsky, M. H. & Sinha, S. Widespread evidence of cooperative DNA binding by transcription factors in Drosophila development. *Nucleic Acids Res.* **41**, 8237–8252 (2013).
12. Ishihama, A. Prokaryotic genome regulation: multifactor promoters, multitarget regulators and hierarchic networks. *FEMS Microbiol. Rev.* **34**, 628–645 (2010).
13. Rydenfelt, M., Garcia, H. G., Iii, R. S. C. & Phillips, R. The Influence of Promoter Architectures and Regulatory Motifs on Gene Expression in Escherichia coli. *PLOS ONE* **9**, e114347 (2014).
14. Gertz, J. & Cohen, B. A. Environment-specific combinatorial cis-regulation in synthetic promoters. *Mol. Syst. Biol.* **5**, 244 (2009).
15. Yu, T. C. *et al.* Multiplexed characterization of rationally designed promoter architectures deconstructs combinatorial logic for IPTG-inducible systems. *Nat. Commun.* **12**, 325 (2021).
16. Urtecho, G. *et al.* Genome-wide Functional Characterization of Escherichia coli Promoters and Regulatory Elements Responsible for their Function. 2020.01.04.894907 Preprint at https://doi.org/10.1101/2020.01.04.894907 (2020).
17. Tewhey, R. *et al.* Direct Identification of Hundreds of Expression-Modulating Variants using a Multiplexed Reporter Assay. *Cell* **165**, 1519–1529 (2016).
18. Johns, N. I. *et al.* Metagenomic mining of regulatory elements enables programmable species-selective gene expression. *Nat. Methods* **15**, 323–329 (2018).
19. Cuperus, J. T. *et al.* Deep learning of the regulatory grammar of yeast 5′ untranslated regions from 500,000 random sequences. *Genome Res.* **27**, 2015–2024 (2017).
20. Van Brempt, M. *et al.* Predictive design of sigma factor-specific promoters. *Nat. Commun.* **11**, 5822 (2020).
21. Cazier, A. P. & Blazeck, J. Advances in promoter engineering: Novel applications and predefined transcriptional control. *Biotechnol. J.* **16**, 2100239 (2021).
22. Blazeck, J. & Alper, H. S. Promoter engineering: Recent advances in controlling transcription at the most fundamental level. *Biotechnol. J.* **8**, 46–58 (2013).
23. Cox III, R. S., Surette, M. G. & Elowitz, M. B. Programming gene expression with combinatorial promoters. *Mol. Syst. Biol.* **3**, 145 (2007).

24. Inukai, S., Kock, K. H. & Bulyk, M. L. Transcription factor–DNA binding: beyond binding site motifs. *Curr. Opin. Genet. Dev.* **43**, 110–119 (2017).
25. Mogno, I., Kwasnieski, J. C. & Cohen, B. A. Massively parallel synthetic promoter assays reveal the in vivo effects of binding site variants. *Genome Res.* **23**, 1908–1915 (2013).
26. Wang, Y. *et al.* Construction of Synthetic Promoters by Assembling the Sigma Factor Binding −35 and −10 Boxes. *Biotechnol. J.* **14**, 1800298 (2019).
27. Stewart, A. J., Hannenhalli, S. & Plotkin, J. B. Why Transcription Factor Binding Sites Are Ten Nucleotides Long. *Genetics* **192**, 973–985 (2012).
28. Guazzaroni, M.-E. & Silva-Rocha, R. Expanding the Logic of Bacterial Promoters Using Engineered Overlapping Operators for Global Regulators. *ACS Synth. Biol.* **3**, 666–675 (2014).
29. Amores, G. R., Guazzaroni, M.-E. & Silva-Rocha, R. Engineering Synthetic cis-Regulatory Elements for Simultaneous Recognition of Three Transcriptional Factors in Bacteria. *ACS Synth. Biol.* **4**, 1287–1294 (2015).
30. Monteiro, L. M. O., Sanches-Medeiros, A., Westmann, C. A. & Silva-Rocha, R. Modulating Fis and IHF binding specificity, crosstalk and regulatory logic through the engineering of complex promoters. 614396 Preprint at https://doi.org/10.1101/614396 (2019).
31. Kotopka, B. J. & Smolke, C. D. Model-driven generation of artificial yeast promoters. *Nat. Commun.* **11**, 2113 (2020).
32. Sample, P. J. *et al.* Human 5′ UTR design and variant effect prediction from a massively parallel translation assay. *Nat. Biotechnol.* **37**, 803–809 (2019).
33. Taskiran, I. I., Spanier, K. I., Christiaens, V., Mauduit, D. & Aerts, S. Cell type directed design of synthetic enhancers. 2022.07.26.501466 Preprint at https://doi.org/10.1101/2022.07.26.501466 (2022).
34. Seo, E., Choi, Y.-N., Shin, Y. R., Kim, D. & Lee, J. W. Design of synthetic promoters for cyanobacteria with generative deep-learning model. *Nucleic Acids Res.* gkad451 (2023) doi:10.1093/nar/gkad451.
35. Gosai, S. J. *et al.* Machine-guided design of synthetic cell type-specific cis-regulatory elements. 2023.08.08.552077 Preprint at https://doi.org/10.1101/2023.08.08.552077 (2023).
36. Nikolados, E.-M. & Oyarzún, D. A. Deep learning for optimization of protein expression. *Curr. Opin. Biotechnol.* **81**, 102941 (2023).
37. Sanabria, M., Hirsch, J. & Poetsch, A. R. Distinguishing word identity and sequence context in DNA language models. 2023.07.11.548593 Preprint at https://doi.org/10.1101/2023.07.11.548593 (2023).
38. Zhang, P. *et al.* Deep flanking sequence engineering for efficient promoter design using DeepSEED. *Nat. Commun.* **14**, 6309 (2023).
39. Garey, M. R. & Johnson, D. S. *Computers and Intractability: a guide to the theory of NP-Completeness*. (Freeman, 1985).
40. Gunawan, A., Lau, H. C. & Vansteenwegen, P. Orienteering Problem: A survey of recent variants, solution approaches and applications. *Eur. J. Oper. Res.* **255**, 315–332 (2016).
41. Maier, D. & Storer, J. A. *A Note on the Complexity of the Superstring Problem*. (1977).
42. Turner, J. S. Approximation algorithms for the shortest common superstring problem. *Inf. Comput.* **83**, 1–20 (1989).
43. Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. Optimization by Simulated Annealing. *Science* **220**, 671–680 (1983).
44. Google. OR-Tools, v9.7. (2023).
45. Golden, B. L., Levy, L. & Vohra, R. The orienteering problem. *Nav. Res. Logist. NRL* **34**, 307–318 (1987).
46. Miller, C. E., Tucker, A. W. & Zemlin, R. A. Integer Programming Formulation of Traveling Salesman Problems. *J. ACM* **7**, 326–329 (1960).
47. Forrest, J. *et al.* coin-or/Cbc: Release releases/2.10.10. (2023) doi:10.5281/zenodo.7843975.
48. Bestuzheva, K. *et al.* Global Optimization of Mixed-Integer Nonlinear Programs with SCIP 8. (2023) doi:10.48550/ARXIV.2301.00587.

49. Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. (2023).
50. OEIS Foundation Inc. Entry A000522. *The On-Line Encyclopedia of Integer Sequences* https://oeis.org/A000522 (2023).
51. Tierrafría, V. H. *et al.* RegulonDB 11.0: Comprehensive high-throughput datasets on transcriptional regulation in Escherichia coli K-12. *Microb. Genomics* **8**, 000833 (2022).
52. Christofides, N. *Worst-case analysis of a new heuristic for the travelling salesman problem*. (1976).
53. van Bevern, R. & Slugina, V. A. A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem. *Hist. Math.* **53**, 118–127 (2020).
54. Сердюков, А. И. О некоторых экстремальных обходах в графах. *Дискретный Анализ И Исследование Операций* 76–79 (1978).
55. 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* 317–330 (Association for Computing Machinery, 2022). doi:10.1145/3519935.3520001.
56. Ilie, L. & Popescu, C. The Shortest Common Superstring Problem and Viral Genome Compression. *Fundam. Informaticae* **73**, 153–164 (2006).
57. Storer, J. A. *Data compression: methods and theory*. (Computer Science Press, Inc., 1987).