Optimizing module matching for synthetic gene circuit design automation

Linh Huynh
Department of Computer Science
& UC Davis Genome Center
University of California, Davis
huynh@ucdavis.edu

Ilias Tagkopoulos
Department of Computer Science
& UC Davis Genome Center
University of California, Davis
itagkopoulos@ucdavis.edu

1. INTRODUCTION

An integral challenge in automated synthetic circuit design is to select the optimal set of parts to populate an abstract circuit topology, so that the circuit behavior best approximates the desired one. In some cases, it is also possible to reuse multi-part constructs, or *modules* that have been already built and experimentally characterized. Efficient part and module matching algorithms are essential to systematically search the solution space and their significance will only increase in the following years due to the projected explosion in part libraries and circuit complexity. Here, we present a matching method based on an iterative node traversal algorithm that can guarantee optimality in the set selection. Results from the integration of the proposed method in the SBROME CAD tool show scalable performance and optimal set selection in a database of experimentally constructed parts and modules. This work represents a fundamental departure from the previous heuristic-based part matching methods [1][2][3][4][5] and is a step towards maximizing efficiency in synthetic circuit design.

2. AN ILLUSTRATIVE EXAMPLE

Assume that we would like to design a circuit that has the topology specified in figure 1b based on an available module library, as shown in figure 1a. Since the final gate is an AND gate and the circuit output is *gfp*, there are only module 4 and module 5 that can match. If we use a greedy strategy based on the matched module size, then the module 4 will be chosen over module 5 (since the first match both A2 and N2) and the gate A1 can only match with the one in module 6, while there are three ways to match the gates N1 and O1. In the first case, where N1 and O1 are matched with module 1, this ultimately does not constitute a solution as there is no way to choose the common input y since module 1 and module 6 share no input. In the second case, N1 and O1 can be matched with module 2 and y can be set to be aTc. However this still does not lead to a solution, since module 2 uses cI as a signal and it will interfere with cI that connects gates A1 (matched with module 6) and N2 (matched with module 4). In the third and last scenario, if N1 and O1 are matched by two modules 8 and 7 as in figure 1c then this actually constitutes a solution since we can choose the common input y as aTc and there is no cross-talk. However this greedy selection requires a final construct with four modules, and it only presents a local optimum. For example, if the output *gfp* is matched with the module 5, then we need only two more modules (module 1 is matched with gates N1 and O1 while module 3 is matched with gates A1 and N2)

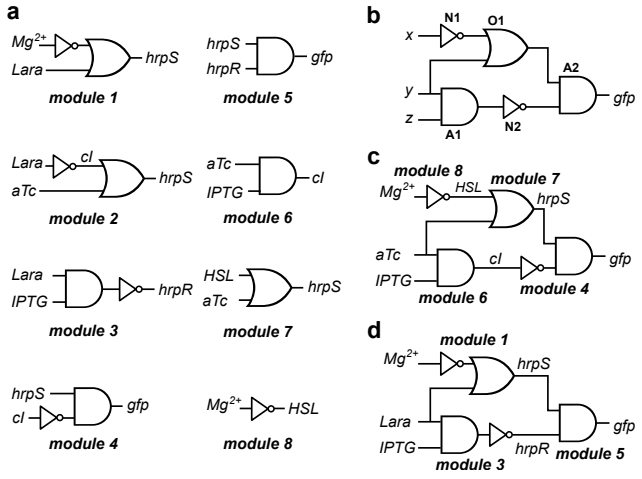

Figure 1: An illustrated example of the module matching problem. (a) The module library. (b) The abstract input circuit graph. (c) A non-optimal matching. (d) An optimal matching

to assemble the whole circuit as in figure 1d. This example illustrates why traditional heuristics are sub-optimal in module matching and it highlights two important features of biological circuits that have to be taken into account, namely the part compatibility and cross-talk.

3. METHODS

Gene circuit representation: Each circuit is represented by a *circuit graph* $G = (V, E, V_I, V_O, \tau_1, \tau_2)$ where V and E are the vertex set and the edge set, V_I and V_O are subsets of V that represent inputs and outputs respectively, τ_1 and τ_2 are vectors that describe the type of each node (ligand, mRNA, protein, gate) and each edge (activatory or inhibitory), respectively.

Problem formulation: Given a circuit graph and a library that contains parts and modules, find a minimum set of parts/modules that can be assembled together to match with the given input circuit graph so the final graph contains no cross-talk between components. In this research, we limit our study for only single output directed acyclic graphs.

A dynamic programming based algorithm: Module matching is similar to topology mapping [6] in electric circuit design which can be solved efficiently by using the dynamic programming approach. However, in contrast to electrical circuits, biological circuits do not have a universal carrier

Table 1: Running time (seconds) for sample circuits

	Module library size		
Design	25	50	97
`not_and_1`	0.01	0.05	0.06
`not_and_2`	0.02	0.28	10.64
`2_to_1_mux_1`	0.02	0.16	0.18
`2_to_1_mux_2`	0.06	0.14	0.15
`2_to_1_mux_3`	0.02	0.54	19.81

of information and they have to utilize multiple molecular species to do so, since there is no explicit wiring to isolate components. This constraint breaks optimal substructure condition of any dynamic programming approach since the optimal solution sometimes has to be constructed from non-optimal sub-solutions because the optimal sub-solutions may be not be compatible or have cross-talk effects.

To overcome these obstacles, we have to store both non-optimal solutions and the information about the connection compatibility and cross-talk effects to ensure the optimal solution is found. For each node v, the *transitive fan-in subgraph* of v is a subgraph that contains v itself and all nodes connected to v by some directed paths. A *matching at* v is a matching of a set of parts and modules with the transitive fan-in subgraph of v that satisfies the constraints of connection compatibility and cross-talk effects. The set $\mathcal{R}(v)$ stores all possible matchings at v and their necessarily related information. In particular, each element $r \in \mathcal{R}(v)$ is a 4-component vector in which i) COST is the number of parts and modules, ii) SIGNAL contains all internal molecular species inside parts and modules that are used to match with the transitive fan-in subgraph of v iii) OUTPUT is the output signal at v with this matching, and iv) information about the recursion so that the procedure TRACEBACK can trace back to find the final solution.

The key idea here is that the set $\mathcal{R}(v)$ will be constructed recursively following algorithm 1. All nodes are traversed by their topological order. At each node v, $\mathcal{R}(v)$ stores all matching combinations that satisfy the constraints of compatibility and cross-talk (checked by procedures COMP and CROSS respectively), each of these combinations contains a matching m at v together the matching at each input of m recursively. This algorithm can be thought of as an exhaustive search that investigates all possible solutions systematically but it's performance is improved by eliminating solutions that violate the constraints of the connection compatibility and cross-talk effects as early as they occur.

Algorithm 1 Module matching algorithm

Input: $G = (V, E, V_I, \{v_o\}, \tau_1, \tau_2)$
  Module & part library $\mathcal{L}$
Output: List of fixed circuits *Sol*
for each node $v \in V$ in the topological order **do**
  $\mathcal{R}(v) \leftarrow \emptyset$
  $\mathcal{M} \leftarrow \text{MATCH}(v, \mathcal{L})$
  for each match $m \in \mathcal{M}$ **do**
    $\mathcal{I} = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \leftarrow \text{INPUT}(m)$
    for $r = (r_{i_1}, r_{i_2}, \ldots, r_{i_k}) \in \prod_{j=1}^{k} \mathcal{R}(v_{i_j})$ **do**
      if $(\text{COMPATIBLE}(m, r) \wedge \neg\text{CROSS}(m, r))$ **then**
        $R(v) \leftarrow R(v) \cup \{\big(\sum_{j=1}^{k} \text{COST}(r_{i_j}) + 1,$
        $\bigcup_{j=1}^{k} \text{SIGNAL}(r_{i_j}) \cup \text{SIGNAL}(m),$
        $\text{OUTPUT}(m),$
        $(m, r_{i_1}, r_{i_2}, \ldots, r_{i_k})\big)\}$
      end if
    end for
  end for
end for
$\mathcal{O} = \{r \in \mathcal{R}(v_o) \mid \text{COST(r) } \textit{is miminum}\}$
$Sol = \text{TRACEBACK}(\mathcal{O})$

4. RESULTS

We built a library of totally 97 parts and modules where 47 are reported in literature and 50 other are synthetic modules (similar constructs with alternative promoters). Since the algorithm will always return the optimal solution, we only consider the running time to evaluate its performance. The running time for each circuit with different library sizes is reported in table 1. In general, the running time scales well in the case where the input topology contains logical gates only (`not_and_1`, `2_to_1_mux_1` and `2_to_1_mux_2`) because their solution space is small and designs are eliminated due to compatibility constraints. When the input topology contains both logic gates and elementary parts, the algorithm has more flexibility to choose smaller modules to match, which leads to an extended solution space and running time (`not_and_2` and `2_to_1_mux_3`).

5. DISCUSSION

We present a module matching strategy that returns the optimal set of parts for synthetic gene circuit design automation. There are several extensions of this work that warrant further investigation. First, we can improve the computational performance by reducing the number of records in each $\mathcal{R}(v)$. This would require a two-step strategy. First, we would have to apply a branch-and-bound technique that is guided by heuristics to find an initial set of solutions, which will allow the pruning of the records with lower score that cannot lead to an optimal solution. Second, we can extend this approach for general graphs that may contain loops or multiple outputs. More far-reaching and ambitious extensions of this method include its application in multi-cellular system design, integration with information flow algorithms for gene networks [7] and population-level simulation tools [8][9][10] for synthetic biology applications [11][12]. As synthetic circuits will continue to grow in size and complexity, methods like the one presented here will constitute the algorithmic basis for future design automation efforts.

6. REFERENCES

[1] L. Huynh, A. Tsoukalas, M. Köppe, and I. Tagkopoulos, "Sbrome: A scalable optimization and module matching framework for automated biosystems design," *ACS Synthetic Biology*, 2012.

[2] L. Huynh, J. Kececioglu, and I. Tagkopoulos, "Automated design of synthetic gene circuits through linear approximation and mixed integer optimization,"

[3] L. Hunyh and I. Tagkopoulos, "A robust, library-based, optimization-driven method for automatic gene circuit design," in *Computational Advances in Bio and Medical Sciences (ICCABS), 2012 IEEE 2nd International Conference on*, pp. 1–6, IEEE, 2012.

[4] L. Huynh, J. Kececioglu, M. Köppe, and I. Tagkopoulos, "Automatic design of synthetic gene circuits through mixed integer non-linear programming," *PloS one*, vol. 7, no. 4, p. e35529, 2012.

[5] F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal, "Automated selection of synthetic biology parts for genetic regulatory networks," *ACS Synthetic Biology*, vol. 1, no. 8, pp. 332–344, 2012.

[6] K. Keutzer, "Dagon: technology binding and local optimization by dag matching," in *DAC*, pp. 617–624, ACM, 1988.

[7] A. Pavlogiannis, V. Mozhayskiy, and I. Tagkopoulos, "A flood-based information flow analysis and network minimization method for gene regulatory networks," *BMC bioinformatics*, vol. 14, no. 1, p. 137, 2013.

[8] V. Mozhayskiy and I. Tagkopoulos, "Guided evolution of in silico microbial populations in complex environments accelerates evolutionary rates through a step-wise adaptation," *BMC bioinformatics*, vol. 13, no. Suppl 10, p. S10, 2012.

[9] V. Mozhayskiy and I. Tagkopoulos, "Horizontal gene transfer dynamics and distribution of fitness effects during microbial in silico evolution," *BMC bioinformatics*, vol. 13, no. Suppl 10, p. S13, 2012.

[10] I. Tagkopoulos, Y.-C. Liu, and S. Tavazoie, "Predictive behavior within microbial genetic networks," *science*, vol. 320, no. 5881, pp. 1313–1317, 2008.

[11] M. Dragosits, D. Nicklas, and I. Tagkopoulos, "A synthetic biology approach to self-regulatory recombinant protein production in escherichia coli," *J Biol Eng*, vol. 6, no. 2, 2012.

[12] I. Tagkopoulos, "Microbial factories under control," 2013.