Coarse Grained Networks Final Report

John Tobin and Dylan Laurianti
Department of Computer Science
Grinnell College

***Abstract*— In the world of network science, and especially with respect to network dynamics, it is typically easier and faster to work on smaller networks with fewer nodes than it is to work on larger ones. This becomes especially important when examining real-world networks, which are often complex with many nodes. In our research, we have investigated a coarse-graining approach to reducing these networks while minimizing loss by preserving as much dynamical behavior as possible. Our method involves examining loss across many partitions of varied networks to find what makes good partitions good, so that we can identify ways to quickly make good partitions of complex networks. If successful, this approach will significantly decrease the time it takes to run dynamical processes on networks, which allows for previously impractical simulations of network dynamics to become reasonable, and therefore more powerful as tools.**

***Index Terms*— Coarse Graining, Network Science, Dynamical Systems**

I. INTRODUCTION

A network, or graph, is a collection of nodes and edges that can be used to represent a wide array of real world situations and problems. For example, there could exist a network where the nodes represent people, and where there is an edge between two people if they have interacted recently. Network dynamics refers to the practice of running dynamical systems on networks to see how certain conditions may propagate through the network over time. For example, the SIS model (Susceptible-Infected-Susceptible model) can be used to model the spread of a disease across a population. If we were to run the SIS model on the aforementioned network representing people and recent interactions, we could obtain information about how a disease would spread across our population. This is a useful tool because researchers could modify the network in many ways and test the model on it with each modification to inform decisions on how a real-world community should change its behavior to minimize infection. There are many of these dynamical models, each useful for modeling different events across different types of networks. However, this type of experimentation on dynamical behavior of networks is often impractical because of the size and complexity of real-world networks. Real world networks may have millions or billions of nodes. Because the time complexity of dynamical systems on networks is bound by the number of nodes in the network, methods to reduce the number of nodes in the network while not substantially changing the dynamical behavior of the network would make many previously impractical network dynamics problems much more practical. We have worked towards developing such methods through our research into coarse-graining networks. Coarse-graining describes the process of reducing a network by creating partitions that map several nodes in the original network to supernodes in a reduced version of the network. These supernodes maintain all edges that any of the original nodes that constitute them had to external nodes in the original network. It must be noted that because of the diversity both of networks and of dynamical systems, there does not exist a general approach to reducing networks that is optimal across all networks and all dynamical systems. Because of this, our approach to this problem isn't to develop a network reduction method that will work on all networks, but rather to develop a method which uses coarse-graining to investigate many different reductions of networks in order to examine patterns across good reductions. With such a method, researchers can tailor-make a reduction of networks to address their specific problems and dynamical processes.

II. BACKGROUND

A. Network Partitioning Difficulty

A closely related problem to network coarse-graining is the detection of communities in networks. Placing groups of nodes into communities based on close interconnectedness is analogous to partitioning nodes into supernodes. The problems also share a lack of ground truth, where we assess the best community detection methods by their ability to align with metadata and the best coarse-graining methods by their ability to produce low loss under some dynamical model. In both cases, there is no predetermined, correct partitioning, known as the ground truth. Both community detection and network partitioning are subject to a No Free Lunch theorem: the accuracy of any algorithm over all possible networks is independent of the algorithm itself. Because of this theorem, any effort to produce algorithms to solve the coarse-graining problem must approach only a subset of possible networks.

Another difficulty in coarse-graining algorithms is computing the fitness of a given partition after an algorithm has created it. Computing the dynamics on a network is equivalent to numerically solving a system of ordinary differential equations with variables equal to the number of nodes, which is complexity $O(s * d_{max}^2 * n)$ on the number

of time steps computed, the maximum degree of a node, and the number of nodes. This makes computation of the dynamics of large networks expensive.

Compounding the difficulty of computing the fitness of partitions is the rate at which the partition space grows with the number of nodes. The number of partitions possible from n nodes to k nodes is given by the Stirling numbers of the second kind, $S(n,k) = \sum_{j=0}^{k} (-1)^j {}_k C_j (k-j)^n$. This rapid growth makes computing the full space of partitions for a given network infeasible, so we rely on sampling the space. Although with larger networks we capture a smaller percentage of the overall space, the smallest unit of discrete change - moving a node from one supernode to another - becomes a less significant change to the overall dynamics of the network as the number of nodes increases. Still, because of the many dimensions of the problem and the complexity of generating good solutions, partitioning methods are generally heuristic.

B. Literature Survey

Papers read in the course of this research focus on two topics, network partitioning and combinatorial optimization. Knowledge about network partitioning allows us both to code algorithms to generate partitions with useful properties and to understand different qualities to look for in partitions generated randomly. Combinatorial optimization will enable us to take our random sample of partitions and both search within that space for minimal loss partitions and refine good partitions further by examining neighbor partitions that weren't sampled.

The most basic foundation of partitioning is the graph bisection, divided the graph into two groups to optimize some property of the bisection. An example of this is the min-cut, which bisects a graph with the minimum amount of edges in between the two components. Partitioning graphs for the purpose of coarse-graining creates many more components so that much of the network is preserved, and then compresses those components into individual supernodes. Because of this, the algorithms we implemented are exclusively bottom-up partitioning methods. Top-down partitioning methods recursively partition the graph into smaller and smaller components. Our bottom-up partitioning methods search for tightly connected clusters and combine individual nodes into supernodes. The two partitioning algorithms that our code implements are spectral clustering and agglomeration reduction. Spectral clustering computes the eigenvalues of the adjacency matrix and then does a k-means clustering using the eigenvalues. Agglomeration reduction is an iterative, greedy approach which collapses the two most closely connected nodes into one until the network is down to the required number of nodes. Other bottom-up algorithms that could be implemented in the future include: simulated annealing, the Louvain algorithm, and random walk algorithms.

An alternative approach to generating good partitions is through combinatorial optimization on the partition space. Our code implements three methods of this, greedy optimization within the sample space, greedy optimization within the entire space, and genetic optimization. Greedy optimizations examine the neighborhood around a starting point and then iterate on the minimum point in that neighborhood until a local minimum is reached. Genetic optimization takes a sample of partitions and then reproduces, crosses, and mutates those partitions for a set number of generations based on a fitness value calculated for each partition. Genetic optimization is advantaged over simple greedy optimization because it can tunnel between local minimums and discover locally optimal partitions that may have otherwise not been reached. It is also much more computationally feasible on partitions with many nodes where even small neighborhoods may require large quantities of calculation to determine the minima. Future optimization methods that could be implemented include simulated annealing and the tabu search algorithm.

III. METHODS

A. Julia Programming Language

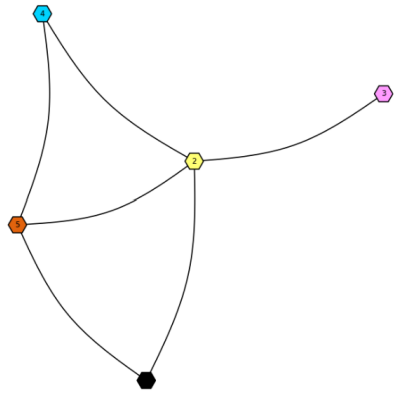
In our research, we are using the Julia programming language because it is comparable in speed to low-level languages like C while maintaining much of the power that high level languages like Python provide. There is even a common expression among the Julia community when espousing its benefits proclaiming that Julia looks like Python, feels like lisp, and runs like C/Fortran. This is to say that despite the fact that Julia is a powerful high-level language, it approaches the speed and efficiency of C in many cases. Julia was a particularly good choice for this research because we had some Python code that was used in the early stages of proposing the Coarse Grained Networks project. This code was written by Professor Nicole Eikmeier alongside several collaborators, and it provided a strong starting point for our research. Because the code was written in Python, it was natural to translate it to Julia since both languages are syntactically similar. Translating the code made the existing operations faster, while also allowing us to build off of it entirely in Julia, resulting in an efficient repository of code for our research. Beyond those reasons, Julia also has a substantial array of libraries and tools for network science making it a great choice for our network science research.

B. Matrix Representation of Networks

Representing networks with adjacency matrices is both convenient and efficient. In an adjacency matrix representation of a network, a matrix of size $n \times n$ is used where n is the number of nodes in the network. For any element (i, j) (where i is a column number and j is a row number) of the matrix, if the value of $(i, j) = 0$, then there is no edge between node i and node j. If the value of $(i, j) \neq 0$, then there is an edge from node j to node i with weight equal to the value of (i, j). Undirected networks are symmetrical across the diagonal, so (i, j) equals (j, i) for all values of

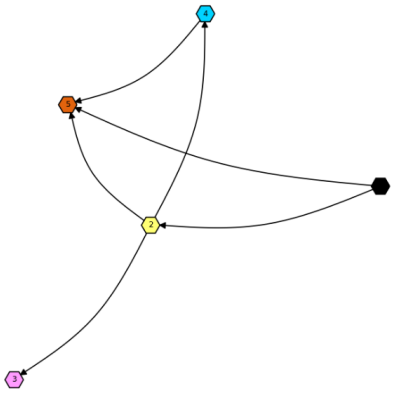
$i, j \leq n$. Below is an example of such an adjacency matrix and the network that it represents:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$



Because the matrix is symmetrical across the diagonal, the network is undirected. A directed version of the same network would look like this:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Since the weights are all 1 in these matrices, we consider both of these networks unweighted.

C. Coarse-Graining

The basis of the Coarse-Graining technique that we are using to reduce networks is graph partitioning. Partitioning describes the process of assigning nodes in the original network to supernodes in the reduced version of the network. Each node in our networks has a node number that is used to refer to these nodes. Our partitioning process makes sure to keep these node numbers referring to the same nodes throughout so that partitions are directly comparable to each other.

We store partitions in dictionaries with key-value pairs where the key is the number of the node in the original network and the value is the number of the node that the original node has been mapped to in the reduced version of the network. Let n be the number of nodes in the original network, and let r be the number of nodes in the reduced version of the network. In each partition dictionary, there are n distinct keys and r distinct values. Because $n > r$, some of the nodes in the reduced network must be matched to more than one node in the original network. These nodes are called "supernodes" because more than one node from the original network has been mapped to them in the reduced network. Supernodes preserve all edges from all nodes in the original network that were mapped to them (excluding edges to other nodes within the same supernode).

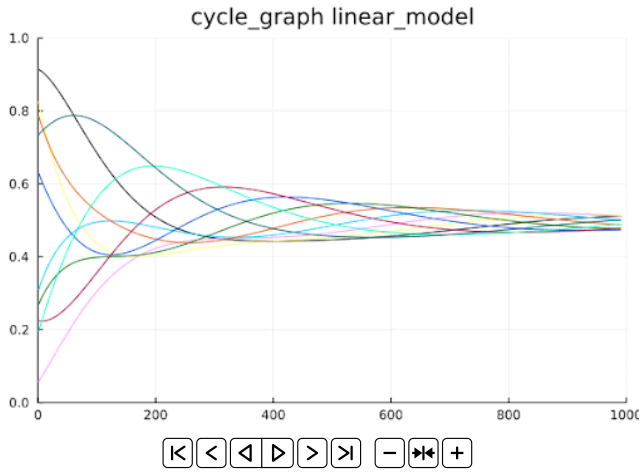
In our research, we're trying to find what makes good partitions good, and in order to do that, we have to compare thousands of partitions. To do this, we take a network that we're interested in partitioning, and then we generate thousands of distinct partitions of that network in the manner described above. Once we have all of these partitions, we need a way to determine which are better than others, and to do this we use a measure of loss that we obtain from dynamical models.

D. Dynamical Models and Loss

Loss is the term we're using to refer to the difference in dynamical behavior between a network and a reduction of that network. We're concentrating on difference in dynamical behavior because our research is focused on reducing networks while maintaining as much dynamical behavior as possible.

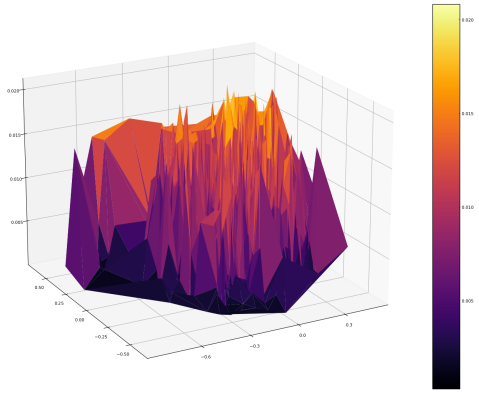
Modeling dynamics on networks uses nodes, edges, and node variables to model the change in a system over time. Mathematically, we use differential equations to let each node's variables be influenced by the nodes with which it shares edges. There are many types of dynamical models. In our testing, we have used linear dynamical models, the SI and SIS models for modeling disease spread, the kuramoto model, the Lotka-Volterra model, and linear/nonlinear opinion dynamics. We're able to test these models on many types of graphs including structured graphs like cycle graphs, line graphs, and stochastic block model graphs, as well as more random graphs like configuration model graphs and GNP graphs generated using the Erdős–Rényi model. We're using many types of models across many types of graphs because our goal is to look for patterns that can help us determine how to partition graphs to minimize loss, and we want to be able to look at many types of graphs and dynamical models to be able to find such patterns more easily.

Below is an example of how node variables spread through a cycle graph over time using the linear dynamical model. Each line represents the value of a node variable, and over time these node variables all converge because in a cycle graph, the entire graph is a cycle, so all nodes are influencing each other.

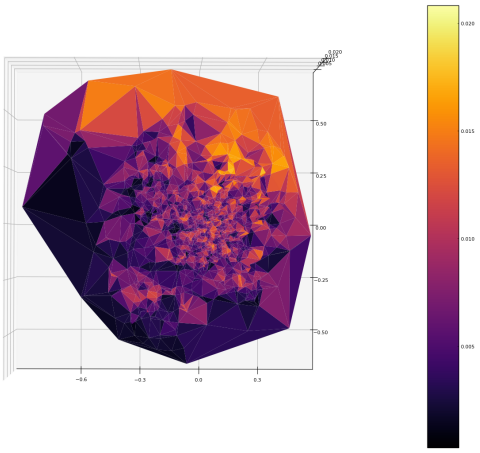


To determine a partition's loss, or the difference in dynamical behavior between the original network and the partitioned network, we run a dynamical model on the original network, then run the same dynamical model on the reduction and compare the results. If we generate many partitions and find the loss for each partition relative to the original network, then we can generate a plot that displays each partition and its loss. This type of plot is called a loss landscape, and it's useful in visualizing an entire partition space so that we can easily identify good and bad partitions. Below is an example of a loss landscape of a GNP network with 25 nodes reduced to 10 nodes across 500 partitions evaluated with linear dynamics.

Side view:

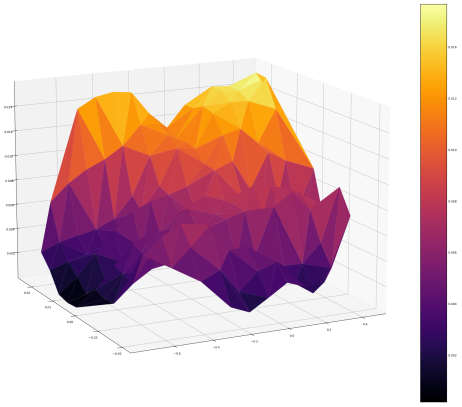


Top-down view:

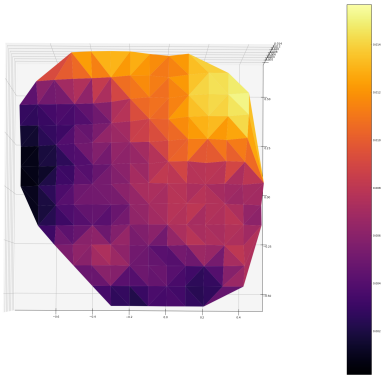


In these plots, each point is a different partition. The x and y values are determined by a function called variation_of_information which calculates the difference between partitions, so similar partitions are close to each other on the x, y plane. The loss of each partition is represented by the z value. areas with relatively low loss are colored purple, and areas with relatively high loss are colored orange/yellow. We also have an R function to smooth the loss landscapes, and using that function on these plots yields the smoothed loss landscapes below.

Side view:



Top-down view:



i97-88=

E. Finding Qualities of Good Partitions

Once we have all of these partitions with their respective loss values, and we've generated a loss landscape, we have lots of information about many partitions and we can quickly identify which are good and which are bad. The problem is that these partitions are very specific to the type of network, type of dynamical model, the sizes of the network and network reduction, and the specific network structure. For this reason, a partition that may be excellent for a particular set of these conditions may lead to enormous loss in a different context. It is for this reason that at this stage in the process, we want to try to identify patterns among good partitions for a variety of loss landscapes, so that hopefully we can find out how to make better methods for creating partitions, as opposed to generating thousands of random partitions.

One approach that we're using to try to find qualities of good partitions is gradient descent optimization. Gradient descent finds local minima on a curve by moving lower along the curve on each step. Our function to do this is called findLocalMinimum and it first chooses a random point on the loss landscape as a starting point (we can optionally specify this starting point). Next, it looks at all other points on the loss landscape within a user-specified radius and finds the partition with the lowest z (loss) value. It then starts the process over again from that partition and keeps going until it reaches a partition for which there are no partitions within the radius that have a lower z value. Our function records each partition that it visits along the path to the local minimum in a CSV file so that we can hopefully identify patterns between the partitions that are chosen along the way.

In addition to using the gradient descent optimization on the sample space, once a decent partition is found and less large moves are expected to be made, greedy optimization on the partition space becomes useful. This allows us to expand past the local optima of the sample space and looks closely around them for the locally best possible partitions.

Another way to find partitions with good qualities is through genetic optimization. Given a sample of partitions, genetic optimization tends to preserve good supernodes, and exploratively combine and mutate partitions until other good supernodes are found. In this way, if one, potentially locally optimal partition has a good coarse-graining solution for a subgraph and another partition has a good solution for another subgraph, these partitions have a chance to be crossed in a way that combines the two good subsolutions each partition contains. Over many generations, we observe that the genetic algorithm combines local minima partitions and tends to produce partitions similar to the best local optima of the space encircled by the original parent population.

IV. EXAMPLE EXPERIMENT

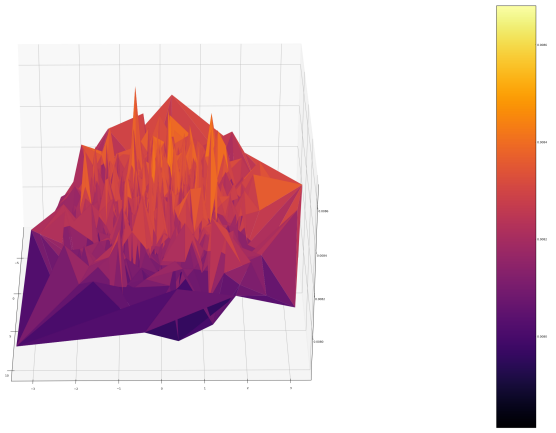
We'll do a small-scale example experiment here so that the process is clear and the methods above are contextualized.

First we must decide what kind of network and dynamical model to use. We have many options for each, but for the purposes of this example, we'll use the SIS model on a Stochastic Block Model network. SBM graphs are a good choice for this type of experimentation because they have a strong underlying structure even when they are randomly generated, so we have a higher chance of getting meaningful data.

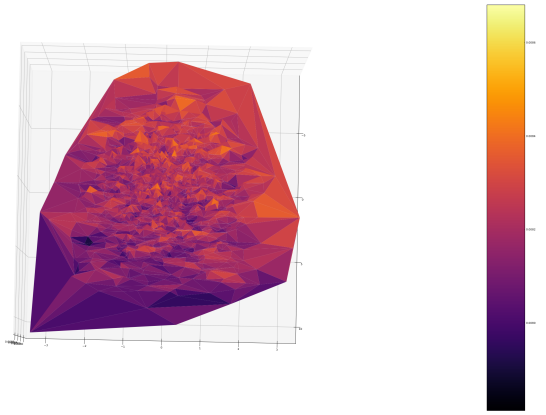
Before running an experiment on a new device, make sure that numCores in Startup.jl is set to the number of CPU threads that your device has. Also make sure that when you open the julia REPL, running Threads.nthreads() returns the number of CPU threads that your device has. If it doesn't, close julia, and open it again with the following invocation: "julia –threads=auto". The first step to conducting an experiment is to open runExperiment.jl in the test folder of CoarseGrained. In this file, we can modify all of the experimental parameters to match the exact kind of experiment we want to run. We'll make sure model is set to SIS_model, NetworkType is set to "SBM", we'll set numOriginalNodes to 200, numReducedNodes to 100, and numPartitions to 2000. Since we're using an SBM, we can ignore DegreeArray and GNPProbability. We'll set cint to 75 and cext to 40, we'll set plot to true, then we'll set the x and y radius to something that seems reasonable like xRadius = 2 and yRadius = 0.6. We'll also keep startingPartition = "" so that the radius-based search begins from a random partition. Now that all of the parameters have been filled in, we can save and run the experiment from the REPL. From the root of the CoarseGrained directory, run the experiment by typing "include("test/runExperiment.jl")".

With these parameters on a computer with 10 CPU cores, the experiment should take around an hour. When it finishes, three files should have been generated. There should be two new files in data/visualization_data. One with x, y, and z values associated with the resultant loss landscape and one that also includes data about the partitions in both dictionary and array form. Another file will have been produced in data/localMin_data that contains the partitions along the path to the local Minimum partition. A plot of the resultant loss landscape is also produced in a separate window, and it looks like this:

Side view:



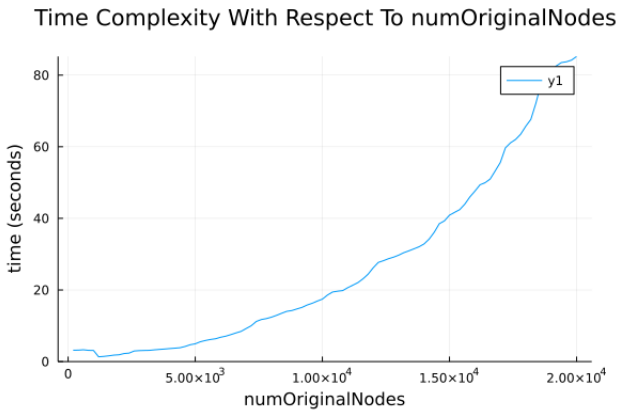
Top-down view:



Now we can also look at the local search data that was generated to see if we can find patterns between partitions that were chosen along the path to a local minimum. It's likely that the first result won't be that interesting, so to run more tests, run LocalSearchTest.jl from runtests.jl several times while modifying the parameters of findLocalMinimum until satisfying results appear.

V. FINDINGS AND ANALYSIS

A. *Time Complexity*



Time complexity appears to be exponential with respect to the number of nodes in the original network. We haven't been able to fully investigate time complexity with respect to the number of nodes in the reduced network or the number of partitions, but from preliminary testing, they also appear to be exponential in terms of time complexity.

VI. CONCLUSIONS

We have not yet found any meaningful patterns among good partitions, but we have much of the framework set up to be able to investigate partitions. We have run several experiments that haven't yielded significant results yet, but we look forward to being able to investigate coarse-grained networks further in the near future.

ACKNOWLEDGMENTS

We would like to thank Professor Nicole Eikmeier for directing this project and mentoring us along the way. We would like to thank Professor Eikmeier's research team, Alice Schwarze, Nicholas Landry, Phil Chodrow, Mari Kawakatsu, and Benji Zusman, that produced the python code off of which we based our Julia code. We would also like to thank the other research team we shared the lab with, Caitlin Abreu, Han Xie, Tanmaie Kailash, and Luke Klein-Collins for their help.

REFERENCES

[1] A. D. Martin and K. M. Quinn, "A Review of Discrete Optimization Algorithms," The Political Methodologist, vol. 7, no. 2.
[2] J. B. Orlin, A. P. Punnen, and A. S. Schulz, "Approximate local search in combinatorial optimization," SSRN Electronic Journal, 2003.
[3] J. Cong and M. L. Smith, "A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI Design," Proceedings of the 30th international on Design automation conference - DAC '93, 1993.
[4] L. Peel, D. B. Larremore, and A. Clauset, "The ground truth about metadata and community detection in networks," Science Advances, vol. 3, no. 5, 2017.
[5] M. A. Webb, J.-Y. Delannoy, and J. J. de Pablo, "Graph-based approach to systematic molecular coarse-graining," Journal of Chemical Theory and Computation, vol. 2019, no. 15, 2018.
[6] M. Toril, I. Molina-Fernández, V. Wille, and C. Walshaw, "Analysis of heuristic graph partitioning methods for the assignment of Packet Control Units in Geran," Wireless Personal Communications, vol. 60, no. 4, pp. 611–633, 2010.
[7] P. Zhang, "Combinatorial optimization problem solution based on improved genetic algorithm," AIP Conference Proceedings, 2017.