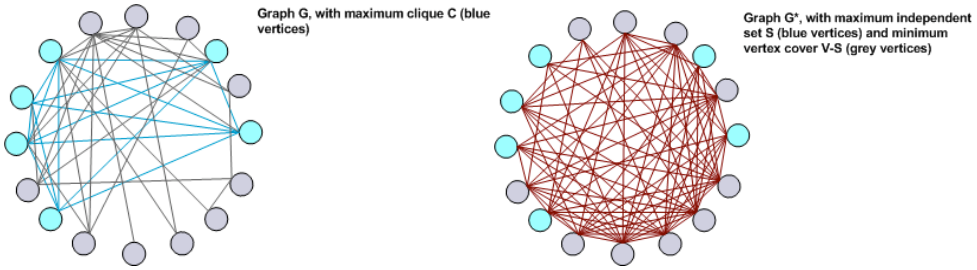
The Maximum Clique Problem

J. Jeffry Howbert (jhowbert), Jacki Roberts (jackirob)
CSEP 521 Applied Algorithms
Winter 2007

Description

Given an undirected graph G = (V, E), a clique S is a subset of V such that for any two elements u, v ∈ S, (u, v) ∈ E. Using the notation E_S to represent the subset of edges which have both endpoints in clique S, the induced graph G_S = (S, E_S) is complete. Finding the largest clique in a graph is an NP-hard problem, called the maximum clique problem (MCP). Cliques are intimately related to vertex covers and independent sets. Given a graph G, and defining E* to be the complement of E, S is a maximum independent set in the complementary graph G* = (V, E*) if and only if S is a maximum clique in G. It follows that V – S is a minimum vertex cover in G*. There is a separate weighted form of MCP that we will not consider further here.



MCP was one of the 21 original NP-hard problems Karp enumerated in 1972. While MCP is closely related to minimum vertex cover, approximation solutions for minimum vertex cover do not translate directly to approximations for MCP. MCP is a canonical example of a highly inapproximable problem, for which there is no polynomial-time approximation that guarantees a constant approximation ratio.

Applications

Traditional application of maximum clique algorithms include:

- patterns in telecommunications traffic
- design of error-correcting codes
- fault diagnosis on large multiprocessor systems
- tiling geometry problems (Keller's conjecture)
- computer vision and pattern recognition

Graph theoretic approaches have also found widespread application in modern biological research. Here we highlight two commonly used and powerful methods that involve detection of cliques in a derived graph.

Matching of three-dimensional molecular structures is vital in at least two areas of biological research: drug discovery and comparative modeling of protein structure (reviewed in Butenko and Wilhelm (2006)). In drug discovery, it often happens that several molecules of apparently unrelated structure are active on the same drug target. If it can be shown that the interatomic distances among a subset of atoms in one drug match the distances among a similar-sized subset in a second drug, one can then postulate a shared pharmacophore which accounts for the interaction of both drugs with the target. With proteins, it is similarly important to know whether a pair of proteins contains similar folding patterns, as this may support hypotheses regarding common evolutionary origins and/or function; however, this is typically hard to determine in an objective manner, as the folding patterns of proteins are highly complex. In both cases, one first constructs a complete edge-weighted graph representing each drug or protein. For drugs, the vertices are typically atoms, and edge weights are interatomic distances. For proteins, the vertices may be secondary structural elements such as α-helices and β-sheets, in addition to atoms, and edge weights may represent angular as well as spatial relationships. Once a pair of molecular graphs is constructed, the problem of interest is to find the maximum common subgraph (MCS) between them, where the MCS is defined as the largest subset of atom/structural element pairs that have matching distances/angles. To find the MCS, the molecular graphs are first converted to a correspondence graph, which identifies *all* graph-to-graph pairs of elements with matching distances/angles. More specifically, for a pair of graphs G_1 = { V_1,

E_1 } and G_2 = { V_2, E_2 }, their correspondence graph C has as V_C = (v_1, v_2) = V_1 X V_2, and two vertices (v_1, v_2) and (v_1', v_2') are connected in C if edge values $w_{v1\text{-}v1'} = w_{v2\text{-}v2'}$ (within a user-defined threshold). Application of a clique-detection algorithm to the correspondence graph then efficiently identifies the MCS. This general approach has been implemented in several widely used software packages, including PROTEP for matching protein structures, and DISCO for finding drug pharmacophores. More recently, it has been extended to the identification of complementary surfaces on proteins, a core piece in the problem of computational docking of biomolecules.

Another use of clique-detection methods is in cluster analysis of gene expression data. Modern DNA microarray technology can generate simultaneous data on the expression levels of thousands of genes in a cell line or tissue, often across dozen of timepoints and hundreds of samples, all run in parallel. Datasets routinely comprise millions of measurements. What biologists hope to extract from such datasets are patterns of coordinated changes in expression, as these may reveal new insights into gene networks involved in normal biology or disease. A typical first step toward this extraction is cluster analysis, which seeks to partition the data into groups so that data points within a group are more similar to one another than to points in other groups. A variety of clustering techniques have been used, such as K-means, neural networks, and hierarchical clustering (reviewed in Jiang, et al (2004)). Among the graph-theoretic approaches is that of Ben-Dor, et al (1999), who introduced the concept of a *corrupted clique graph*. It is assumed the true clusters can be represented by a set of disjunct cliques, whose union is the clique graph H, and that the input dataset deviates from the true cluster structure due to random errors that result from the noisy, complex experimental process. The input data is first converted to a similarity graph G, where vertices represent genes, and an edge exists between any pair of genes whose expression pattern satisfies a criterion of similarity. Edges and non-edges in G are then flipped with probability α until a clique graph is obtained, with the goal of removing the "corruption" from G with as few flips (errors) as possible. When applied to published gene expression datasets, this algorithm (called CAST) recovered clusters similar to those from other methods, and required fewer assumptions by the user. In addition, it was quite fast, running in time $O(n^2[\log n]^c)$.

Because MCP is NP-hard, practical applications on large sets commonly involve the related but computationally simpler problem of maximal cliques: cliques not included in any larger clique. Tosic and Agha (2004) used maximal cliques in the context of coalition formation among autonomous agents to define an algorithm that allows independent unmanned vehicles to locate other agents and team up with them in an ad-hoc manner to share resources. The need with a large number of agents, who may or may not be acting on a common goal, is to break them up into groups that enable them accomplish tasks that may not be possible when working in isolation. For example, information that each agent has about its local environment can be shared to create a broader map. Lightweight agents may not have sufficient resources onboard to accomplish all of their tasks, and may need to find other agents that are not only nearby, but also have capabilities they lack.

When the number of agents grows sufficiently large, it is impractical to have a centralized authority appoint a leader or break the agents into cooperating groups. Tosic and Agha provided a method for allowing the agents themselves to choose their affiliation based on distance and utility metrics, provided they can communicate throughout the group-choosing process. Fault tolerance recommends that all agents in a group be able to communicate, so the maximal clique model is a natural one for defining groups. The fluidity of agents moving into or out of range requires that the group-formation algorithm be efficient, as it may be called multiple times during a task set. The authors' algorithm involves several stages in which an agent first identifies the other agents in its neighborhood, then considers the available groups as each of its neighbors chooses an affiliation. As this information comes in, the agent eventually makes its own choice, and will remain affiliated with that group until the task set is complete.

Isolated but intra-communicating subgroups are also the hallmark of cells of people working together while trying to avoid detection. Hayes (2006) attempted to reverse-engineer the National Security Agency's telephone surveillance program from descriptions of its goals and knowledge about its targets. Early published accounts of the program made references to "call graphs." A terrorist cell needs to keep communication outside the cell minimal to reduce exposure, and communication inside the cell open to increase nimbleness, particularly in the days immediately preceding an attack. While it is impossible to determine precisely the capabilities of the NSA, a maximal clique of small-to-moderate size in a call graph, with little communication outside the clique, might be a plausible flag for a suspicious set of nodes. Studies

by AT&T in the late 1990's studied problems with similar structure, examining the graph that resulted from all of the phone calls across the network in one day. This graph had over 50 million nodes (phone numbers) and 170 million edges (calls between numbers). They used an approximate, probabilistic method to find cliques in the graph. The largest maximal clique they found had 30 vertices, representing 30 people, each of whom talked to all 29 other people in the clique on that day. This experiment revealed a major obstacle in getting useful information out of this analysis, as the analysis discovered more than 14,000 distinct instances of 30-node cliques. Assuming a small number of malicious cells, the large number of false positives would make using cliques as a starting point prohibitively noisy. They could plausibly be used, however, to hone in on a target already under suspicion for other reasons.

Current Research

When analyzing the merits of various exact and approximate methods for finding maximum cliques, it is useful to understand the underlying combinatorial complexity of the problem. In a graph $G = \{ V, E \}$, there are 2^n subsets of V ($n = | V |$). The most naïve approach to finding a maximum clique involves enumerating all 2^n subsets, and testing up to $n(n - 1)/2$ edges in each to determine whether the graph induced by that subset is complete. This approach requires time $O(n^2 2^n)$. Exhaustive enumeration is, of course, unnecessary. For the decision form of the problem (i.e. determine whether a clique of size $\geq k$ exists in G) only those subsets of size k need be enumerated. This reduces the time required to $O(k^2 b(n , k)) = O(k^2 n^k)$ [$b(. , .)$ is a binomial coefficient]. For the related optimization problem, the comparable enumeration has running time upper bounded by $k^2 \cdot \sum_{i=0 \text{ to } k} b(n, i)$, where k is the size of the actual maximum clique in G.

By definition, exact algorithms deliver solutions which are guaranteed correct. Their challenge is to do so while minimizing the portion of the combinatorial search space they must traverse. In the past 30 years, a large number of exact methods have been described (reviewed in Bomze, et al (1999)), many of which are both practical to implement and have useful performance on graphs up to few thousand vertices. Most of these perform a partial enumeration of vertex subsets, using some version of a branch-and-bound strategy to reduce the search space. Several improvements on simple branch-and-bound use heuristic coloring schemes to expand cliques found at intermediate stages of the search. Quadratic programming formulations of the maximum clique problem have also been popular. The merits of greedy vs. non-greedy branching rules were explored by Pardalos and Rodgers.

It is not easy to judge which exact algorithms are best for practical applications. Many publications report performance testing against some of the standard DIMACS graph instances, but head-to-head comparisons of algorithms are rare. One of the simplest branch-and-bound pruning strategies is that of Carraghan and Pardalos (C&P; 1990). In spite of its simplicity, it is still regarded as one of the most efficient exact algorithms, especially on sparse graphs. It was chosen as one of two reference algorithms to beat in the maximum clique section of the Second DIMACS Challenge in 1992-1993 (Johnson and Trick (1996)), and the code for it is still available online at ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/dfmax.c. A synopsis of the C&P algorithm follows.

The first step of C&P is to fix an order of the n vertices of G. In the case of dense graphs, computational times are substantially reduced if this order is in ascending order of vertex degree, i.e. v_1 is the vertex of smallest degree in G, v_2 is the vertex of smallest degree in the subgraph induced by $V - \{ v_1 \}$, v_3 is the vertex of smallest degree in $V - \{ v_1, v_2 \}$, etc. Lower density problems typically run faster if the vertices are left in their original (random) order.

The main search of C&P operates in a recursive fashion. Three principles are used to define the recursions:

- Each v_i has a set of adjacent vertices $A(v_i)$, and the largest clique containing v_i can only involve vertices in $A(v_i)$. This naturally restricts further analysis to the subgraph induced by $v_i + A(v_i)$.
- Any clique involving two vertices v_h and v_j, where v_i follows v_h in the order, will be found when v_h is considered; thus v_h can be eliminated from $A(v_i)$ when v_i is subsequently considered. This further restricts analysis to the subgraph induced by $v_i + (A(v_i) \cap \{ v_{i+1}, v_{i+2}, \ldots, v_n \})$.
- If there is a clique of size k in $v_i + (A(v_i) \cap \{ v_{i+1}, v_{i+2}, \ldots, v_n \})$, there is necessarily a clique of size $k - 1$ in $A(v_i) \cap \{ v_{i+1}, v_{i+2}, \ldots, v_n \}$.

The algorithm begins by spawning a search for the largest clique containing each vertex v_i, from v_1 up to v_n in order. Within the search for a given v_i, the algorithm considers each element in $A(v_i) \cap \{ v_{i+1}, v_{i+2}, \ldots,$

v_n }, and spawns a subsearch for the largest clique containing that element, subject to the same subgraph restriction strategy described above. The spawning of recursions continues until there are no vertices remaining in a subgraph; at this point, a trace of the vertices upon which recursions were spawned defines a clique. Simple bookkeeping identifies the maximum size clique found over the entire search, and the vertices contained within it.

As described thus far, C&P achieves some efficiency by restricting its recursions to progressively smaller subsets of vertices which are known to have some order of proximity in the graph. However, the addition of a simple pruning strategy further reduces the search space, in a dramatic way. Suppose the size of the largest clique found so far is L, and that the algorithm is currently on a branch where j levels of recursion have been launched. At this point, there are j vertices accumulated in the candidate clique. Now imagine the algorithm is considering launching the next level of recursion from vertex v_i in the current subgraph, and there are k vertices left in the current subgraph which it has yet to consider. If $j + k \leq L$, there is no possibility that a clique larger than L can be found by further recursion from v_i or any of the other vertices remaining in the current subgraph. The algorithm therefore immediately abandons this subgraph and returns up a level in the recursion.

More recently, Östergård (2002) published a variant on C&P in which the vertices are also ordered, but the search for the largest clique associated with each vertex proceeds in the opposite sense, i.e. from v_n down to v_1. Because the vertices are ordered, the largest clique found for v_i establishes a lower bound on the size of cliques found for any subsequent vertex v_j, $j < i$. This enables a pruning strategy that appears to be very powerful, as the algorithm was faster than C&P in a head-to-head comparison on over 60 test graphs.

Remarkably, publications on exact methods for maximum clique almost never provide an analysis of algorithmic complexity, even ones like C&P that are based on simple, non-heuristic principles. Tarjan and Trojanowski, and later Robson, published exact algorithms with worst-case complexities of $O(2^{n/3})$ and $O(2^{0.276n})$, respectively, which improve on the $O(2^n)$ required for full enumeration of vertex subsets. However, the algorithms are quite complex, involving consideration of many special cases, and it is not clear either has actually ever been implemented. Wilf (1986) showed that (under certain probabilistic assumptions) the average complexity of a backtracking algorithm is subexponential, growing at rate $O(n^{\log n})$. For C&P, our analysis establishes a (weak) upper bound of $p^2 \sum_{i=1 \text{ to } n} 2^p$, where $p = \min(D_i, n - i)$ and D_i is the degree of vertex v_i. (This does not even take into account the pruning strategy.) In the extreme case where all $D_i = n - 1$ for all i, this sum becomes $O(n^2 2^n)$, but in more general circumstances the sum is upper bounded by $d_m^2 \cdot n 2^{dm}$, where d_m is the largest degree of any vertex. For sparse graphs in particular, this is an obvious dramatic reduction in complexity.

Research into approximate methods (also reviewed in Bomze, et. al. (1999)) focuses on ways to efficiently solve problems on general graphs, and generally relies on the comparisons with the DIMACS benchmarks to establish validity. Historically, the approximation algorithms involve a greedy local search in which vertices are added or deleted from a candidate set. Recent algorithms improve on this model, as they determine a neighborhood of vertices incident to an initial maximal clique is found, and consider adding the vertices in this neighborhood in turn, possibly deleting vertices from the initial set. This works under the assumption that once a good set is found, another good set is probably nearby. Such local algorithms need to explicitly avoid cycling on the same sets repeatedly. Most algorithms address this need by randomly perturbing the current set and by performing multiple runs. Neighborhood searches grow quickly in complexity as the neighborhood size increases, so much research is spent on choosing neighborhoods and searches wisely. The results are by definition local optima.

Hansen, Mladenović, and Urošević (2004) present a variable neighborhood search (VNS) algorithm based on this standard form. A generic VNS algorithm involves a series of neighborhood changes interleaved with local searches, until a specified stopping condition is met, usually a plateau in improvement or CPU elapsed time. The notion of a metric is needed to compare distances between solutions when augmenting the neighborhoods.

The authors move back and forth between cliques in G and independent sets/vertex covers in G* (the complement of G) because of their underlying equivalence.

A generic VNS involves first an initialization step: choosing a finite value k_{max}, which is often 1, predefining k_{max} neighborhoods N_1, …, N_{kmax}, and choosing initial optimal clique set X (which may be empty). Then the following procedure is repeated until the stopping condition is met:

- Set k=1
- Repeat until k=k_{max}:
  - (Shaking) Select a random solution X' in N_k(X). This prevents cycling.
  - (Local Search) Apply a greedy local search method to X' to find local optimum X''
  - (Move option) If X'' is better than current optimum X, Set X=X'' and k=1. Else k=k+1

For this implementation the authors define the solution space to be the set of vertex covers in G*. If we use S to represent a clique in G, T=V-S is a vertex cover for G*, and is therefore a member of the solution space. The distance between two solutions is the symmetric difference of the sets, so $d(T_i,T_j)=|T_i-T_j| + |T_j-T_i|$. For a given solution, the notation $N_m(T)$ represents the set of solutions at distance m from T. An initial solution is found, from which the neighborhood set is defined. From this point, the algorithm attempts to add nodes in the local search phase, drops and add nodes if the optimum does not improve on the current best set, or drops nodes in order to reset the current neighborhood entirely.

The local search heuristic is the crucial determinant of an algorithm's efficiency. The principal components of the local search are the greedy selection algorithm, the rule used to break ties between nodes of equal value, and the rule used to determine how to handle a plateau in the current neighborhood. We consider the first of these here, though the article addressed a non-random tie-breaking rule as well.

The initial solution X is defined by a node with no edges in the complementary graph G*, as this node has edges to all vertices in G. If no such node exists, vertices with nondecreasing numbers of edges in G* are examined in turn, and one with the minimum number chosen at random. One then decides which vertex to consider next, the greedy part of the local search. The authors examined several options for this step: (1) take the vertex with maximum order in G, (2) take the vertex with maximum order in G*, (3) use a random number to decide whether to use (1) or (2), or (4) choose any vertex randomly that is neither in the current clique nor its related vertex cover.

Few approximation articles include complexity analysis, and focus instead on speed of computation and the size of the largest clique detected when run in the DIMACS benchmark. The state-of-the-art at the time of the article's writing, in terms of both speed and clique size found, was reactive local search (RLS). While no particular greedy option out of the four the authors considered stood out on all tests, the results overall were within 1 vertex of the maximum clique size found by RLS, and did not significantly exceed the time RLS required except on a few pathological test cases.

Other Perspectives

Johan Håstad showed in 1999 that there is no polynomial time approximation algorithm that is within a ratio $n^{1-\epsilon}$ of the optimum for any $\epsilon>0$ unless NP=ZPP[1], where n=|V|. Subhash Khot improved this bound in 2001 to $n/(2^{(\log n)^{1-\gamma}})$. Approximation algorithms using neighborhood local search procedures seem to dominate much of the recent research. Algorithms inspired by other sciences attempt to find global, rather than local, optima. These include neural networks, genetic algorithms, simulated annealing (from lattice structures found in condensed-matter physics), and DNA computing.

Bibliography

1. A. Ben-Dor, R. Shamir, Z. Yakhini, "Clustering Gene Expression Patterns," Journal of Computational Biology, 1990.
2. I. Bomze, M. Budinich, P. Pardalos, M. Pelillo, "The Maximum Clique Problem," Handbook of Combinatorial Optimization, 1999.
3. S. Butenko, W.E. Wilhelm, "Clique-detection Models in Computational Biochemistry and Genomics," European Journal of Operational Research, 2006.
4. R. Carraghan, P.M. Pardalos, "An Exact Algorithm for the Maximum Clique Problem," Operations Research Letters 9, 1990.
5. P. Hansen, N. Mladenović, D. Urošević, "Variable Neighborhood Search for the Maximum Clique," Discrete Applied Mathematics, 2004.
6. B. Hayes, "Connecting the Dots," American Scientist, Vol. 94, Sept/Oct 2006.
7. D. Jiang, C. Tang, A. Zhang, "Cluster Analysis for Gene Expression Data: A Survey," IEEE Transactions on Knowledge and Data Engineering, 2004.

[1] ZPP is the class of problems for which a polynomial-time, randomized algorithm exists that guarantees the correct answer.

8. D.S. Johnson, M.A. Trick, Eds., DIMACS Series, Volume 26: Cliques, Coloring, and Satisfiability, American Mathematical Society, Providence, RI, 1996.
9. P.R.J. Östergård, "A Fast Algorithm for the Maximum Clique Problem," Discrete Applied Mathematics, 2002.
10. P. Tosic, G. Agha, "Maximal Clique Based Distributed Group Formations for Autonomous Agent Coalitions," In Coalitions and Teams Workshop, AAMAS, 2004.