On Effectively Finding Maximal Quasi-Cliques in Graphs*

Mauro Brunato[1], Holger H. Hoos[2], and Roberto Battiti[1]

[1] Dipartimento di Ingegneria e Scienza dell'Informazione,
Università di Trento, Trento, Italy
brunato|battiti@disi.unitn.it
[2] Department of Computer Science,
University of British Columbia, Vancouver, BC, Canada
hoos@bs.ubc.ca

Abstract. The problem of finding a *maximum clique* in a graph is prototypical for many clustering and similarity problems; however, in many real-world scenarios, the classical problem of finding a *complete* subgraph needs to be relaxed to finding an *almost complete* subgraph, a so-called *quasi-clique*. In this work, we demonstrate how two previously existing definitions of *quasi-cliques* can be unified and how the resulting, more general quasi-clique finding problem can be solved by extending two state-of-the-art stochastic local search algorithms for the classical maximum clique problem. Preliminary results for these algorithms applied to both, artificial and real-world problem instances demonstrate the usefulness of the new quasi-clique definition and the effectiveness of our algorithms.

1 Introduction

Finding maximum cliques, *i.e.*, largest complete subgraphs, within a given graph is a well-known NP-hard combinatorial problem that is particularly intractable because of its non-approximability. However, state-of-the-art heuristic search methods, in particular stochastic local search algorithms, are typically able to find maximum or near-maximum cliques surprisingly effectively.

In real-world scenarios, relationships commonly represented by graphs are often subject to noise, resulting in erroneously missing or added edges. This motivates generalisations of the maximum clique problem in which the objective is to find maximum size subgraphs that are almost fully connected — so-called *quasi-cliques*.

* Holger Hoos acknowledges support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Discovery Grant 238788-05; Mauro Brunato and Roberto Battiti acknowledge support by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

Definition 1. *Given an undirected graph* (V, E)*, and two parameters* λ *and* γ *with* $0 \leq \lambda \leq \gamma \leq 1$*, the subgraph induced by a subset of the node set* $V' \subseteq V$ *is a* (λ, γ)-quasi-clique *if, and only if, the following two conditions hold:*

$$\forall v \in V' : \quad \deg_{V'}(v) \geq \lambda \cdot (|V'| - 1) \tag{1}$$

$$|E'| \geq \gamma \cdot \binom{|V'|}{2}, \tag{2}$$

where $E' = E \cap (V' \times V')$ *and* $\deg_{V'}(v)$ *is the number of elements of* V' *connected to* v*.*

Note that for $\lambda = \gamma = 1$, classical cliques are obtained; consequently, the problem of finding a maximum (λ, γ)-quasi-clique for arbitrary λ, γ is at least as hard as the maximum clique problem.

Condition (1) enforces a lower bound on the degree of each node within the quasi-clique, while condition (2) poses a lower bound on the overall number of edges. While both constraints have been previously proposed in the literature, to the best of our knowledge they have not previously been combined. In this work we propose this combination and show that by using it, cluster detection for noisy data (as motivated above) can be improved.

The remainder of this work is organised as follows: Section 2 discusses the motivation of our research and places it in the context of related work; Section 3 outlines the two maximum clique finding techniques we chose to adapt to our new problem formulation; Section 4 discusses the modifications that these algorithms require to operate in the more difficult space of quasi cliques; and Section 5 presents an initial experimental study on graphs that have been constructed to capture important characteristics of several real-world applications.

2 Context and Related Work

Relevant examples of quasi-clique applications and related clustering approaches include classifying molecular sequences in genome projects by using a linkage graph of their pairwise similarities [1], analysis of massive telecommunication data sets [2], as well as various data mining and graph mining applications, such as cross-market customer segmentation [3].

Different clustering techniques capable of identifying significant interconnected sub-structures in the presence of random noise have been used recently to analyze complex interconnected networks ranging from autonomous systems in the Internet, protein-protein interaction networks, e-mail and web-of-trust networks, co-authorship networks, and trade relationships among countries [4–7]. In the area of data mining, Du et al. have recently studied techniques to enumerate all maximal cliques in a complex network [8].

This contribution aims at extending the work done by the authors in efficient clique algorithms, in particular Reactive Local Search (RLS) and Dynamic Local Search for Maximum Clique (DLS-MC). Both algorithms are based on stochastic local search methods. The DLS-MC algorithm for the maximum clique problem

is based on the idea of assigning penalties to nodes that are selected to be part of a clique [9]. The RLS algorithm uses a reactive mechanism to control the amount of diversification during the search process by means of prohibitions [10, 11]. Both algorithms are outlined in Sec. 3. The objectives of our work are as follows:

- To develop efficient and effective heuristic algorithms for the problem of finding maximum (λ, γ)-quasi-cliques.
- To understand which heuristic components are effective as a function of the relevant graph parameters.
- To understand the effect of the problem parameters (average and minimum connectivity requirements) on the empirical run-time of the algorithms and on the quality of the results they produce.
- To assess the viability of the proposed techniques to discover a wide range of maximal quasi-cliques. Discovering many comparable solutions is critical when a quasi-clique detection module is only a first step towards a compact description of a graph, or when additional requirements or constraints are given by the user in a later phase to select from a large set of quasi-cliques.

3 Clique finding heuristics

This section outlines the two state-of-the-art stochastic local search algorithms for the maximum clique problem whose extension to quasi-clique finding is described later in this paper. A description of the data structures used by both algorithms precedes their outlines.

3.1 Data structures for classical clique search

Both, DLS-MC and RLS use two data structures to efficiently compute local search steps from a current clique, $V' \subseteq V$: the set of nodes $\mathrm{Add}(V')$ that can be added to the current clique V' (i.e., of nodes in $V \setminus V'$ that are connected to all nodes in V') and the set $\mathrm{Miss}(V')$ of nodes in V that are connected to all nodes but one in the current clique V', as shown in Fig. 1. Note that $\mathrm{Add}(V')$ is called PossibleAdd in the original RLS description, and *improving neighbour set* in the DLS-MC paper. The set $\mathrm{Miss}(V')$ is called OneMissing in RLS and *level neighbour set* in DLS-MC. Based on these sets, the following types of search steps can be efficiently implemented:

- **Add one node:** Once the node $v \in \mathrm{Add}(V')$ to be added has been chosen, all elements of $\mathrm{Miss}(V')$ not connected to v are dropped; elements of $\mathrm{Add}(V')$ not connected to v are moved to $\mathrm{Miss}(V')$.
- **Remove one node:** All nodes $v \in V'$ are eligible for removal. Once a node v to be removed has been chosen, it is added to $\mathrm{Add}(V')$; furthermore, all nodes in $\mathrm{Miss}(V')$ that are not connected to v are promoted to $\mathrm{Add}(V')$. The set of nodes from $V \setminus (V' \cup \mathrm{Miss}(V') \cup \mathrm{Add}(V'))$ that are *not* connected to v is then scanned to identify nodes to be added to $\mathrm{Miss}(V')$.

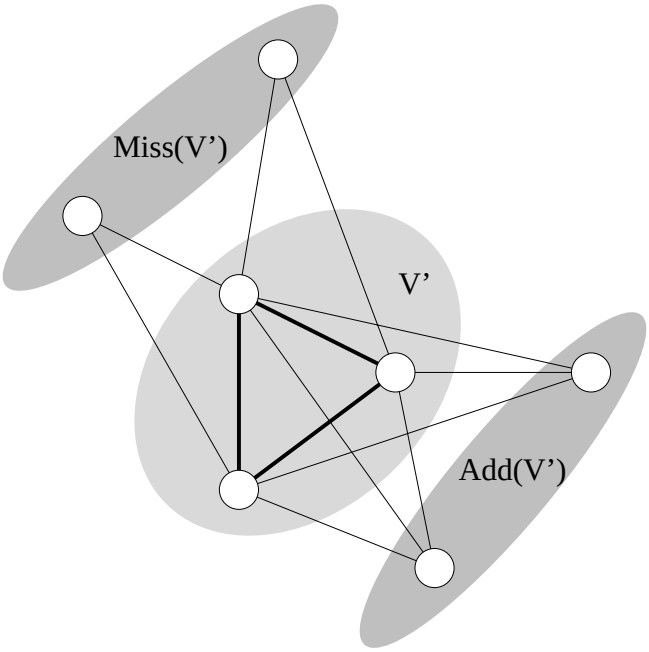


Fig. 1. Node sets involved in a classical clique search.

– **Plateau move:** A node $v \in \text{Miss}(V')$ is added to the current clique, followed by the removal of the node $v' \in V'$ that is not connected to v; the sets $\text{Add}(V')$ and $\text{Miss}(V')$ are incrementally updated similarly as in the case of add and remove moves.

All three types of moves result in a clique.

3.2 Reactive Local Search (RLS)

Reactive Local Search (RLS) [10, 11] operates by maintaining the current clique V' and modifying it with two basic moves: node addition and node removal. The basic diversification mechanism of RLS is based on Tabu Search [12, 13]: every time a node is added to or removed from the current clique, it cannot be considered for removal or addition (it is *prohibited*) for the next T moves. The basic Tabu Search heuristic is complemented by a memory-based reactive scheme that automatically modifies the parameter T in order to adapt it to the problem instance. Important details of RLS are as follows:

Choice of the best move. At every step, the algorithm looks for a node to be added to the current clique V' among all non-prohibited nodes in $\text{Add}(V')$. A node $v \in \text{Add}(V')$ with a maximal number of connections within $\text{Add}(V')$ (*i.e.*,

one whose addition to the clique leaves as many candidates as possible in the next step) is chosen, with ties broken randomly. If no nodes can be added, a non-prohibited node $v \in V'$ whose removal results in the largest set $\mathrm{Add}(V' \setminus \{v\})$ (containing the candidates for addition to be considered in the following step) is chosen for removal, with ties broken randomly. If all candidates for addition and removal are prohibited, then a node is removed uniformly at random. In all cases, the selected node is prohibited for the following T steps.

Memory reaction. Different graphs require different degrees of diversification, so a reactive mechanism is used in RLS to adjust the value of T. All visited cliques are mapped to the step of their last appearance as current cliques (using a hash table for fast retrieval). If the current clique has been visited too recently, the prohibition period T is increased. If a predefined number of steps occur without any increase of T, meaning that no clique is repeated too early, then T is decreased.

Restart mechanism. If the algorithm performs a given number of steps without improving the size of the maximum clique, the search is restarted by resetting the current clique to an unused node of the graph that is chosen uniformly at random. If all nodes have already been used at least once, then the random choice is made from the set of all nodes. All other parameters and data structures are reset to their initial values ($T \leftarrow 1$, the hash table is emptied).

3.3 Dynamic Local Search for Maximum Clique (DLS-MC)

Dynamic Local Search for Maximum Clique (DLS-MC) [9] uses two basic types of search steps to modify the current clique V': node addition and plateau moves.

The basic diversification mechanism of DLS-MC is based on *penalty values* associated with each node. At the beginning of the search process, the current clique is set to a single node that is uniformly chosen at random and all node penalties are set to zero. The algorithm then alternates between two search phases:

- In the *expansion phase*, which continues as long as $\mathrm{Add}(V')$ is not empty, a node from $\mathrm{Add}(V')$ with minimum penalty is selected (with ties broken randomly) and added to the current clique;
- In the *plateau phase*, which continues as long as $\mathrm{Add}(V')$ is empty and $\mathrm{Miss}(V')$ contains at least one node, a node in $\mathrm{Miss}(V')$ with minimum penalty is selected for addition, and the node of V' not connected to it is removed from the current clique. Moreover, at the beginning of the plateau phase, the current clique is recorded (as V''), and the phase is terminated when $V' \cap V'' = \emptyset$.

In addition, a prohibition mechanism is used to prevent the plateau phase from cycling through a small set of cliques: once a node is chosen for addition, it becomes prohibited until the end of the current plateau phase. At the end of the plateau phase, two actions are taken:

Penalty update. The penalty values of all nodes in the current clique V' are increased by 1. Additionally, every pd update cycles, all nonzero penalties are decremented by 1, so that penalties are 'forgotten' over time.

Clique perturbation. If $pd = 1$ (meaning that every increase in penalties is immediately cancelled, so that penalties always remain equal to zero), a new current clique is generated by adding a new node $v \in V$ chosen uniformly at random and removing from the current clique V' all nodes that are not connected to v. If, on the other hand, $pd > 1$ (penalties are used), then the current clique is reduced to the last node that was added to it.

4 Supporting data structures for quasi-clique search

When adapting DLS-MC and RLS to the quasi-clique setting, more complex sets of nodes must be maintained in order to support the basic search steps.

The number of edges is always an integer, therefore we can conveniently rewrite constraints (1) and (2) in order to use integer variables to store clique bounds:

$$\forall v \in V' : \quad \deg_{V'}(v) \geq \lceil \lambda \cdot (|V'| - 1) \rceil \tag{3}$$

$$|E'| \geq \left\lceil \gamma \cdot \binom{|V'|}{2} \right\rceil \tag{4}$$

4.1 Adding one node

Let us define the set of *critical* nodes in a (λ, γ)-clique as those nodes whose degree in V' is high enough to justify their presence, but would fail to satisfy condition (3) if the clique size increased without adding an edge to them:

$$\mathrm{Crit}(V') := \{v \in V' : \deg_{V'}(v) < \lceil \lambda \cdot |V'| \rceil\}.$$

Consider, for example, the case shown in Fig. 2, where $V' = \{1, 5, 6, 7\}$. It is easy to verify that V' is a (λ, γ)-quasi-clique for $\lambda = \gamma = 2/3$. If, however, a new node were added to V', the degree of nodes 1 and 7 would no longer satisfy condition (3), unless edges are added to both of them; therefore, $\mathrm{Crit}(V') = \{1, 7\}$.

The addition of a node should also satisfy the global density constraint (4), so any new node must contribute at least $d_{V'}$ edges to the clique, where

$$d_{V'} := \left\lceil \gamma \cdot \binom{|V'| + 1}{2} \right\rceil - |E'|$$

is the minimum number of edges that must be added in order to maintain constraint (4). In the example from Fig. 2, a new node must contribute at least $d'_V = 2$ edges. Consequently, a node $v \in V \setminus V'$ is eligible for addition to V' if the three following conditions hold:

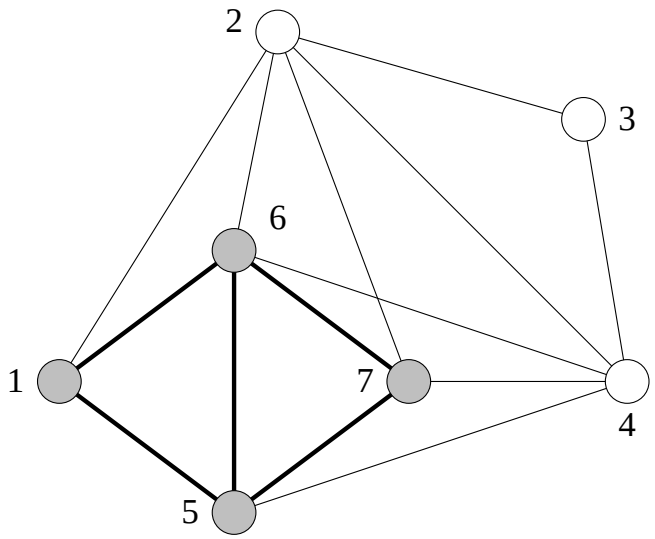


Fig. 2. A graph and a $(2/3, 2/3)$-quasi-clique (shaded nodes).

- v has an adequate degree in V' according to (3);
- v has enough edges to nodes in V' that as a result of adding it, the edge density in $V' \cup \{v\}$ does not fall below threshold (4);
- all critical nodes in V' receive at least one more edge when adding v (i.e., v is connected to all critical nodes).

Based on these conditions, the set of nodes eligible for addition to V' is defined as

$$\mathrm{Add}(V') := \left\{ v \in V \backslash V' : \deg_{V'}(v) \geq \max\{\lceil \lambda \cdot |V'| \rceil, d_{V'}\} \wedge \{v\} \times \mathrm{Crit}(V') \subseteq E \right\}.$$

In the example from Fig. 2, the only eligible node (connected to all critical nodes, and contributing at least 2 edges) is node 2.

4.2 Removing one node

To be eligible for removal from a (λ, γ)-clique, a node must not be connected by an edge to any *removal-critical* node, where the set of removal-critical nodes is defined as follows:

$$\mathrm{RCrit}(V') := \{v \in V' : \deg_{V'}(v) - 1 < \lceil \lambda \cdot (|V'| - 2) \rceil\}.$$

By losing an edge, such nodes would no longer satisfy constraint (3) for the resulting, smaller quasi-clique. In the example from Fig. 2, nodes 1 and 7 are removal-critical, and therefore nodes connected to them, i.e., 5 and 6, cannot be removed.

Secondly, if a node has sufficiently high degree in V', its removal would cause the global edge density to fall below threshold (4). The maximum number of edges that can be removed from quasi-clique V' without violating the global density constraint is

$$e_{V'} := |E'| - \left\lceil \gamma \cdot \binom{|V'|-1}{2} \right\rceil .$$

In the example from Fig. 2, up to 3 edges can be removed.

Generally, the set of edges that are eligible for removal is therefore

$$\mathrm{Rem}(V') := \left\{ v \in V' : \big(\{v\} \times \mathrm{RCrit}(V')\big) \cap E = \emptyset \wedge \deg_{V'}(v) \leq e_{V'} \right\}.$$

In the example, only nodes 1 and 7 can be removed, while removing node 5 or 6 would leave nodes 1 and 7 with too small a degree to remain within the clique.

4.3 Plateau moves

In the classical clique case, a plateau move is a node removal followed by a node addition, and the clique property is maintained throughout the process. Note that removals do not maintain the (λ, γ)-clique property in the general case, but the property could be restored after a node addition; therefore, a plateau move in the quasi-clique domain must be regarded as atomic with respect to the quasi-clique constraints. Despite this atomicity from the quasi-clique's point of view, the removal and addition operations must be performed sequentially in one of the two possible orders. In this work, we only consider the "add, then remove" option; however, the opposite is possible.

We allow a node $v \in V \setminus V'$ to be added to the quasi-clique in the context of a plateau move if, and only if, it is subsequently possible to identify at least one node in V' whose removal would maintain or restore the quasi-clique property. Considering that a node different from v should be removed, the nodes eligible for addition only need to be sufficiently well-connected in V':

$$\mathrm{PAdd}(V') := \{v \in V \setminus V' : \deg_{V'}(v) \geq \lambda \cdot (|V'| - 1)\}.$$

The set of nodes eligible for removal depends on the added node. Note that, in general, $\mathrm{PAdd}_{V'} \supseteq \mathrm{Add}_{V'}$, and once $w \in \mathrm{PAdd}_{V'}$ is chosen, $V' \cup \{w\}$ is not necessarily a (λ, γ)-clique. In order to select the node to be removed, we define a *plateau-critical* set of nodes, depending on V' and on w:

$$\mathrm{PCrit}(V', w) := \{v \in V' \cup \{w\} : deg_{V'}(v) - 1 < \lambda \cdot (|V'| - 1) \wedge (v, w) \notin E\}.$$

This set contains the nodes that would not satisfy the quasi-clique property when removing an edge, unless they receive an additional edge when node w is added. Note that w itself may not belong to this set if its degree is not high enough.

When choosing a node to be removed, we must make sure that it is not connected to a plateau-critical node, and that we don't remove too many edges

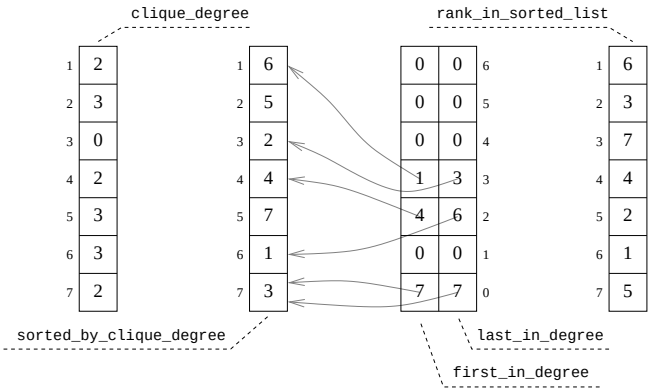


Fig. 3. Data structures used to support quasi-clique search, instantiated for the quasi-clique from Fig. 2.

from $V' \cup \{w\}$. The maximum number of edges we can afford to lose from $V' \cup \{w\}$ in order to guarantee its quasi-clique property is

$$r_{V',w} := |E'| + \deg_{V'}(w) - \gamma \cdot \binom{|V'|}{2}.$$

Considering this, the set of candidates for removal is defined as

$$\mathrm{PRem}(V', w) := \left\{ v \in V' : \deg_{V' \cup \{w\}}(v) \leq r_{V',w} \wedge (\{v\} \times \mathrm{PCrit}(V', w)) \cap E = \emptyset \right\},$$

and the only nodes that can actually be added to the quasi-clique in the plateau step are those whose corresponding set of candidates for removal is not empty.

4.4 Support set implementation

The previously defined support sets can be maintained efficiently using the array structures illustrated in Fig. 3. In particular, we make use of an array `sorted_by_clique_degree`, where nodes $v \in V$ are sorted by increasing $\deg_{V'}(v)$. Everytime the clique is updated by adding or removing a node, the V''-degree of some nodes will be increased or decreased, and the array must be modified to reflect the new ordering. In order to perform these changes in time $O(\deg_{V'}(v))$, where V' is the clique before updating and v is the node to be added, we maintain three additional arrays:

`rank_in_sorted_list`
: the inverse of `sorted_by_clique_degree`;

`first_in_degree`
: whose i-th element contains the first index in `sorted_by_clique_degree` containing a node with the given degree in the current quasi-clique;

`last_in_degree`
whose i-th element contains the last index in `sorted_by_clique_degree` containing a node with the given degree in the current quasi-clique.

Note that we assume that all arrays are indexed starting from 1, with the exception of `first_in_degree` and `last_in_degree`, which are indexed by node degrees and therefore start from zero.

In general, it is not possible to incrementally maintain $\mathrm{Add}(V')$ and $\mathrm{Rem}(V')$ in the same way as for the classical maximum clique problem. For instance, $\mathrm{Add}_{V'}$ is monotonic for greedy clique constructions, while for (λ, γ)-cliques, this set can acquire or lose elements at every step. Using the previously mentioned arrays, however, it is possible to restrict the search of eligible nodes to smaller sets of candidate nodes. This is effective, since all conditions for node removal or addition are defined on clique degree ranges; in particular, critical and removal-critical nodes typically fall into a small range of degrees within the current quasi-clique, usually just one or two.

5 Experimental analysis

In the following we demonstrate how by using our new quasi-clique definition, based on the combined constraint set (1)+(2) with proper non-zero settings of the local and global density parameter, and our generalised versions of RLS and DLS-MC, we can effectively find densely connected subgraphs that are not accessible to classical clique finding approaches.

5.1 Identification of overlapping communities

Following the motivating considerations on quasi-clique usage for community identification, a set of benchmark graphs has been designed and generated in order to capture some fundamental features of communities. The generated graphs depend on five parameters: the number of nodes N, number of communities G, intra-community link probability P_{in}, inter-community link probability P_{out} and fraction f of overlap between communities. The last parameter, f, controls the overlap between communities as follows: If N is the number of nodes in our graph and G divides N, we define G groups $g_0, \ldots, g_{G-1}$ so that groups g_i and g_{i+1} share $f \cdot N/G$ nodes. To generate such a graph, a random permutation $\pi : \{0, N-1\} \mapsto \{0, N-1\}$ is generated. For every node $n \in \{0, \ldots, N-1\}$ and every group index $i \in \{0, \ldots, G-1\}$, node n belongs to group g_i if and only if

$$\frac{N}{G} \cdot \left(i - \frac{f}{2}\right) \leq \pi(n) < \frac{N}{G} \cdot \left(i + 1 + \frac{f}{2}\right).$$

Note that if $f = 0$ the G groups form a partition of the node set, while if $f > 1$ overlapping regions extend to non-adjacent groups. Once assignments of nodes to groups are computed, with the permutation function acting as a "scrambling factor," edges are assigned to each pair of nodes according to the number of

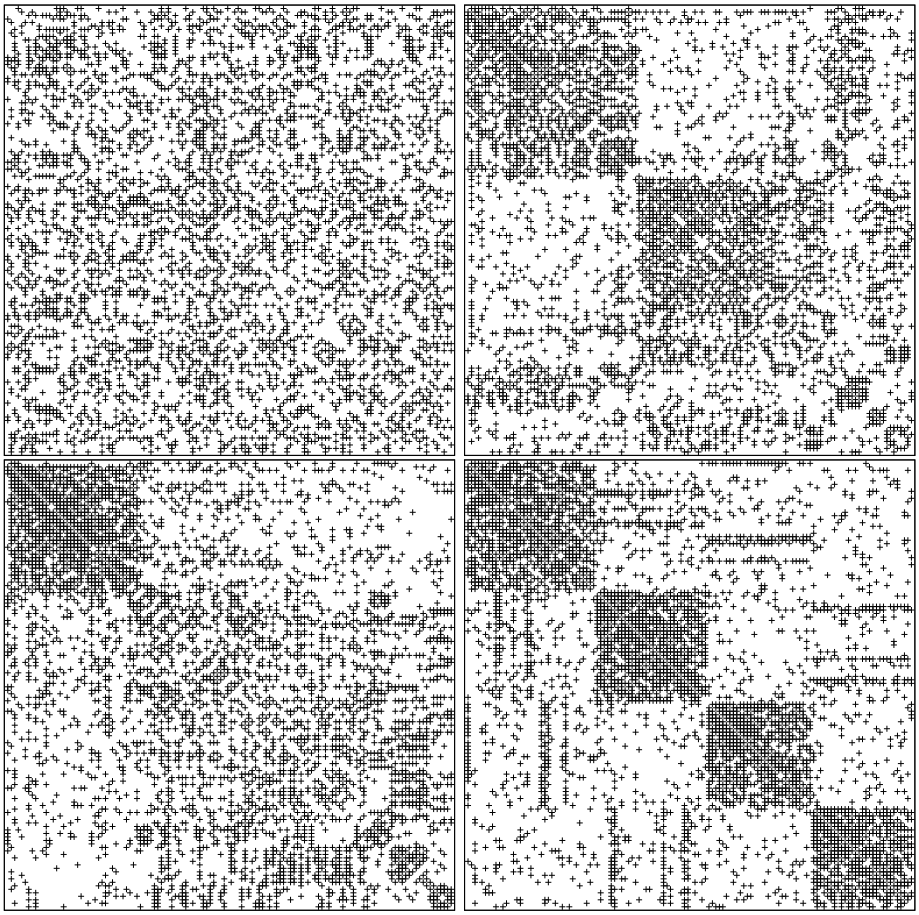


Fig. 4. Adjacency plot of overlapping community graph with nodes sorted by order of appearance in the identified quasi-cliques according to different values of (λ, γ); from top left to bottom right: original ordering, $(0, 0.4)$, $(0.65, 0)$ and $(0.65, 0.4)$. Black crosses represent edges.

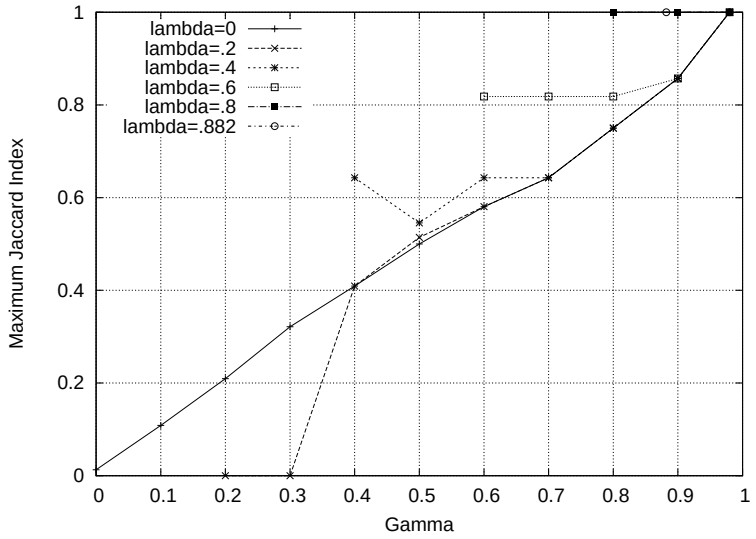


Fig. 5. Differences between maximum (λ, γ)-quasi-cliques in a noisy version of a co-authorship graph and the maximum clique in the original graph (for details, see text).

groups that they have in common. Let h_{ij} represent the number of groups in common between nodes i and j, then:

$$\Pr((i,j) \in E) = \begin{cases} P_{\text{out}} & \text{if } h_{ij} = 0 \\ 1 - (1 - P_{\text{in}})^{h_{ij}} & \text{otherwise.} \end{cases}$$

Figure 4 shows an example with $N = 128$ nodes, numbered from 1 to 128, that are distributed in $G = 4$ communities with overlap factor $f = 1/8$ (groups are formed by nodes 1–33, 30–65, 62–97, 94–128). The probability of edge (i, j) to appear in the graph is $P_{\text{in}} = 0.7$ if i and j belong to the same community, $P_{\text{out}} = 0.1$ otherwise. Node labels are then randomly permuted, so that there is no longer a correlation between node labels and community membership.

The top left pane of Figure 4 shows the original adjacency matrix: because of the random permutation, no structure is apparent. Next, we generated a list of the largest maximal $(0.4, 0.65)$-quasi-cliques found by a short run of the quasi-clique extension of RLS. When nodes are sorted in the order in which they appear in this list (so that nodes contained in the largest quasi-clique are grouped together, and so on), the resulting adjacency matrix (bottom right pane of Fig. 4) shows a significant structure: nodes belonging to the same community are grouped together. The adoption of constraint (2) alone (top right pane of Fig.4) and of constraint (1) alone (bottom left pane) are not sufficient to correctly identify the four clusters. We also note that communities cannot be directly represented by classical cliques.

5.2 Cliques in noisy graphs

A second experiment considers noisy versions of graphs, in which edges are removed uniformly at random with some small probability. For appropriate values of λ and γ, the maximum clique of the original graph should appear as a (λ, γ)-quasi-clique in the noisy graph. In Figure 5, a co-authorship graph from a scientific database [4] has been processed by removing each edge with 5% probability; subsequently, the DLS-MC algorithm for maximum quasi-clique finding has been executed 10 times for each of many different values of λ and γ. As a measure of similarity between the maximum quasi-clique found by the algorithm and the maximum (classical) clique in the original graph, the maximum value of the Jaccard index over all maximum quasi-cliques found in the 10 runs for each (λ, γ) has been used. (The Jaccard index of two sets U and V is defined as $J(U, V) := |U \cap V|/|U \cup V|$.)

The results of this experiment, shown in Figure 5, indicate that when γ decreases, the maximum quasi-clique in the noisy graph becomes increasingly different from the maximum clique in the original graph. On the other hand, a higher value of λ tends to keep the similarity value higher, while the global constraint alone (curve with $\lambda = 0$) seems to provide worse results. Note, however, that the curve for $\lambda = 0.2$ is an exception, because it performs worse for lower values of γ. This behavior is not fully understood yet.

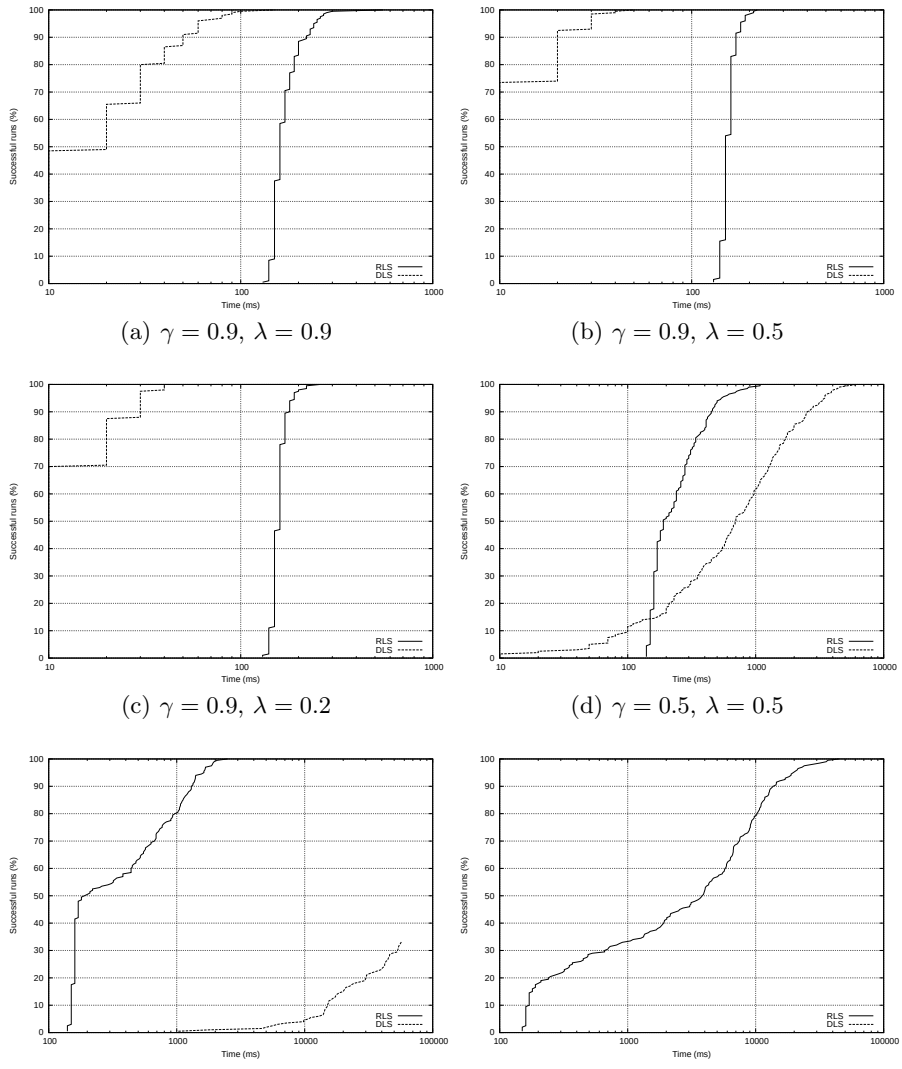
5.3 RLS and DLS-MC run-time comparison

To compare the performance of the previously discussed modifications of RLS and DLS-MC, we ran both algorithms on a graph representing protein-protein interaction data for *S. cerevisiae* (yeast). More specifically, we first determined putative maximum (λ, γ)-quasi-clique sizes for various values of λ and γ by performing long (10-minute) runs of each algorithm. Next, we measured empirical run-time distributions for reaching these target clique sizes based on 200 independent runs per algorithm and quasi-clique parameter setting.

Figure 6 shows the resulting cumulative run-time distribution functions for the two algorithms. As can be seen from these results, for high values of the λ and γ, *i.e.*, for dense quasi-cliques, DLS-MC appears to outperform RLS, while RLS appears to perform better on sparse quasi-cliques. The weak performance of DLS-MC for sparser quasi-cliques seems to be caused by a large number of plateau moves executed by the algorithm, and is currently being further investigated.

6 Conclusions

In this work, a new definition for quasi-cliques has been introduced which combines and generalises previously proposed quasi-clique definitions. In order to generalise high-performance stochastic local search algorithms for the maximum clique problem to the more general problem of finding maximum quasi-cliques, we have introduced new data structures, using which basic local search steps



(a) $\gamma = 0.9$, $\lambda = 0.9$ (b) $\gamma = 0.9$, $\lambda = 0.5$

(c) $\gamma = 0.9$, $\lambda = 0.2$ (d) $\gamma = 0.5$, $\lambda = 0.5$

(e) $\gamma = 0.5$, $\lambda = 0.2$ (f) $\gamma = 0.2$, $\lambda = 0.2$

Fig. 6. Performance comparison between RLS and DLS-MC: percentage of successful runs *vs* run-time for finding putative maximum (λ, γ)-quasi-cliques. Notice that the diagrams have different time scales to accommodate for large variations in performance.

(node addition, removal and plateau moves) can be performed efficiently. Preliminary experiments on artificial and real-world graphs have been presented in order to motivate the adoption of this definition and to demonstrate the effectiveness of our generalised algorithms and their supporting data structures.

In future work, we plan to investigate the effectiveness of our new quasi-clique algorithms in more depth and for a wider range of graphs. It would also be interesting to extend further high-performance stochastic local search algorithms for the maximum clique problem, such as PLS [14], to (λ, γ)-quasi-cliques.

Acknowledgements

We gratefully acknowledge contributions by Srinivas Pasupuleti, who helped during the initial stages of this project, Wayne Pullan, who made his implementation of DLS-MC available to us, and Franco Mascia, whose RLS code for maximum clique finding we used as a basis for implementating our RLS extension for quasi-cliques.

References

1. Matsuda, H., Ishihara, T., Hashimoto, A.: Classifying molecular sequences using a linkage graph with their pairwise similarities. Theoretical Computer Science **210**(2) (1999) 305–325
2. Abello, J., Resende, M., Sudarsky, S.: Massive quasi-clique detection. Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN) (2002) 598–612
3. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. Conference on Knowledge Discovery in Data (2005) 228–238
4. Serrano, M., Boguñá, M.: Clustering in complex networks. I. General formalism. Arxiv preprint cond-mat/0608336 (2006)
5. Hopcroft, J., Khan, O., Kulis, B., Selman, B.: Tracking evolving communities in large linked networks (2004)
6. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**(7043) (2005) 814–818
7. Everett, M., Borgatti, S.: Analyzing clique overlap. Connections **21**(1) (1998) 49–61
8. Du, N., Wu, B., Xu, L., Wang, B., Pei, X.: A Parallel Algorithm for Enumerating All Maximal Cliques in Complex Network. Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on (2006) 320–324
9. Pullan, W., Hoos, H.H.: Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research **25** (2006) 159 – 185
10. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29**(4) (2001) 610–637
11. Battiti, R., Mascia, F.: Reactive local search for maximum clique: a new implementation. Technical Report DIT-07-018, University of Trento (2007)
12. Glover, F.: Tabu search - part i. ORSA Journal on Computing **1**(3) (1989) 190–260
13. Glover, F.: Tabu search - part ii. ORSA Journal on Computing **2**(1) (1990) 4–32
14. Pullan, W.: Phased local search for the maximum clique problem. Journal of Combinatorial Optimization **12**(3) (2006) 303 – 323