Short Paper

Partial k-Plex Enumeration for Feature Filtering and a Novel Application in the Geosciences

Benjamin Parfitt
benjaminlparfitt@gmail.com

Darren Strash
dstrash@hamilton.edu
Department of Computer Science, Hamilton College
Clinton, NY, USA

ABSTRACT

Many optimization problems in AI are solved by transforming the input into a graph and then finding a clique maximizing an objective function. Not only are cliques often too rigid a structure, but the standard method for solving this problem, enumerating *all* maximal cliques and returning one maximizing the objective function, is either too slow or (in dense graphs) infeasible. We show that partial enumeration of k-plexes, via light modifications to clique local search algorithms, is sufficient to find high-quality solutions in both sparse and dense graphs in two example applications.

KEYWORDS

k-plexes, local search, feature filtering, geosciences, applications

1 INTRODUCTION

Cliques—subgraphs in which all vertices are pairwise adjacent—have emerged as a critical structure for solving a variety of problems in artificial intelligence (AI), including such disparate applications as unsupervised feature selection for machine learning (ML) [12, 15], knowledge base queries [3], and object segmentation in video [19, 20]. The problems are first transformed into graphs, which are commonly weighted using problem-specific knowledge. For example, in object segmentation, a clique represents a class of similar objects and disjoint cliques identify multiple classes of dynamic objects in a collection of videos [20].

A custom objective function is often needed to evaluate the quality of potential clique solutions in AI problems. Therefore it is not sufficient to use "out of the box" algorithms for finding a single maximum clique, or even a maximum weight clique; instead, existing algorithms first enumerate all maximal cliques and then return a clique maximizing the objective function.

There are several shortcomings of using cliques (and maximal clique enumeration) to compute solutions to these problems. Firstly, many applications have dense graphs [15, 20], which can contain up to $\Theta(3^{n/3})$ maximal cliques [11] where n is the number of vertices in the graph. In dense graphs (such as those that have 50% of the possible edges), the number of cliques renders clique enumeration infeasible in practice for graphs with even hundreds of vertices, independent of the method. The graphs are often small enough and dense enough that the algorithm by Tomita et al. [18], which has worst-case optimal running time $O(3^{n/3})$, is the best choice in practice. However, we emphasize that even this method is slow for such dense graphs. While it is tempting to use other clique enumeration algorithms from the literature, whether based on the Bron-Kerbosch algorithm [7], or algorithms with bounded delay based on reverse search [2, 5, 10], these are all slower in practice than Tomita et al. [18] and assume the input graph is sparse. Secondly, a clique is often *too rigid* a structure to give the best solution for all instances of a problem. Requiring all edges to be present misses out on potential solutions when graphs are incomplete (due to omissions in data collection) or have errors from flaws in problem formulation.

Instead it would be valuable to consider graph structures that are similar to cliques, but allow for missing edges. In the context of communities, Seidman and Foster [16] introduced one such generalization, called the k-plex, which allows each vertex to have up to $k-1$ non-neighbors in the subgraph. However, k-plex enumeration is even slower than clique enumeration, with the most recent algorithm, called *FaPlexen*, having running time $O(n^2c^n)$ for $c < 2$ [21]. However, local search may be helpful here. Two algorithms, *iterated local search* (ILS) [9] and *phased local search* (PLS) [13, 14] are able to quickly find near-optimal solutions to the maximum clique and maximum weight independent set problems, by making small "local" changes to a current solution that is updated as improved solutions are found. These can easily be modified to keep a list of subgraphs considered.

Our Results. As a proof of concept, we investigate partial enumeration of k-plexes as a substitute for full enumeration of maximal cliques. We empirically observe that enumerating a small subset of k-plexes of a graph using local search can solve instances much faster and even enables finding higher-quality solutions than clique enumeration on some instances. Our prototype algorithm is a light modification of the successful ILS and PLS local search algorithms for cliques. Partial enumeration is able to find solutions in very dense graphs with thousands of nodes that cannot be solved in under 200 hours by clique enumeration. In sparse graphs with thousands of nodes, partial enumeration can even find higher quality solutions than full clique enumeration. Our experiments show that this technique enables feature filtering on graphs of up to 4 280 vertices 79% faster than the method of Schroeder et al. [15] and replaces days of manual work in a novel application in the geosciences.

2 PRELIMINARIES

We consider a simple undirected graph $G = (V, E)$, with a sets of *vertices* $V = \{v_1, v_2, ..., v_n\}$ and *edges* $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We also use *weight functions*, $w : V \to \mathbb{R}^+$, assigning a weight to each vertex, and $w_E : E \to \mathbb{R}^+$, assigning a weight to each edge. Note that we abbreviate an edge $\{v_i, v_j\} \in E$ by $e = v_iv_j$. A graph $H = (V_H, E_H)$ is a *subgraph* of $G = (V, E)$, if $V_H \subseteq V$ and $E_H \subseteq E$. For brevity, we let $n = |V|$, and $m = |E|$. The *neighborhood* of a vertex $v \in V$, denoted $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ is the set of all vertices *adjacent* to v. The *degree* of $v \in V$ is denoted by $\deg(v) = |N_G(v)|$. A *complete graph* is a graph $G = (V, E)$ where $uv \in E$ for all $u, v \in V$, $u \neq v$. A *clique* in a graph $G = (V, E)$ is a subgraph $K = (V_i, E_i)$ that is complete. A *k-plex* is a subgraph

10.48786/alioeuro.2022.14

$L = (V_L, E_L)$ of G, where each vertex has at most $k-1$ non-neighbors in L (i.e., for $v \in V_L$, $|(V_L \setminus N_L(v)) \setminus \{v\}| \leq k-1$).

3 PARTIAL k-PLEX ENUMERATION

As a proof of concept, we combine Phased Local Search (PLS) and Iterated Local Search (ILS) into a simple algorithm to partially enumerate both cliques and non-clique k-plexes. We call this algorithm Phased Local Iterated Search for k-plex Enumeration (PLISkE). Our goal is *simplicity*: we keep the parameters and order of the original phases of PLS [13] and leave optimizations for future work.

Similar to other local search algorithms, PLISkE maintains a growing set of vertices $L \subseteq V$, which in our case is a set of vertices that induces a k-plex for a given value of k. As this set of vertices is updated throughout the algorithm, it is added to a growing set of potential solutions P, which is the partial enumeration of solutions. Our local search first looks for cliques, which are 1-plexes, and then increases k as the search proceeds. In order to allow PLISkE to be applied to a variety of yet unknown problems, we direct our search using an objective function unique to each problem [9], denoting it $f_{\text{Inner}}(L)$. In order to choose the best of all solutions in P we use an additional objective function, $f_{\text{Outer}}(L)$.

Each iteration of PLISkE executes the three phases specified in PLS [13]. Each of our phases consists of three steps: growing L (which we do via a $(1,2)$-SWAP), adding L to our set of potential solutions P, and then perturbing L according to the *vertex selection method* for that phase (which is the same in PLS). The three vertex selection methods are described in more detail below, and are as follows: *uniform random selection*, *penalty selection*, and *degree-based selection*. Each phase runs for 50, 50, and 100 iterations respectively, the same as the original implementation of PLS [13].

In the original (clique) implementation of PLS, two sets of vertices C_0 and C_1 are maintained, where C_i is the set of all vertices having exactly $|L| - i$ neighbors in L. In each phase of PLS, a vertex v from C_0 or C_1 is forced into L, and its non-neighbors are removed from L. This step ensures that L stays a clique and that its size does not decrease. We maintain no such sets here since, as k increases, maintaining vertices with at least $|L| - k$ neighbors in L becomes increasingly expensive, especially in the dense graphs we consider here.

Therefore, unlike PLS, we store no such sets of vertices to force into L. Instead, we begin a phase by growing L if possible via a single $(1,2)$-SWAP, a technique from Andrade et al. [1] that removes one vertex v from L and replaces it with two vertices $u, w \in C = V \setminus L$ while ensuring that L induces a k-plex for the current value of k. Let the running time of f_{Inner} be denoted by $T(f_{\text{Inner}})$. We perform the $(1,2)$-SWAP in $O(T(f_{\text{Inner}}) \cdot n^3 + n^5)$ expected time by, for each of the $O(n^3)$ combinations of $v \in L$ and $u, w \in C$, first checking that $f_{\text{Inner}}((L \setminus \{v\}) \cup \{u, w\}) > f_{\text{Inner}}(L)$, and then confirming in $O(n^2)$ time that the subgraph is a k-plex.

After growing L via a $(1,2)$-SWAP, L is added to P, and then we move on to the perturbation step.

The last step of each phase is to perturb the growing solution by choosing a vertex $v \in C$ using that phase's unique vertex selection criterion, then adding v to L and removing all vertices in L not adjacent to v. Note that this does not guarantee that the vertices in L are all pairwise adjacent (i.e., a clique), but only that they are adjacent to v, ensuring that all subgraphs in P are dense and still k-plexes. We denote this routine in Algorithm 1 by PERTURB(L, Selection Method).

The three vertex selection criteria are as follows:

Algorithm 1 The PLISkE algorithm.

```
proc PLISkE(maxIters, maxSolns)
 1: P = {}                                          ▷ All potential solutions
 2: L = {v_1}                                       ▷ Current growing solution
 3: iters = 0
 4: k = 1
 5: while |P| < maxSolns and iters < maxIters do
 6:     iters = iters + 1
 7:                                                 ▷ Phase 1: Uniform Random
 8:     for 50 iterations do
 9:         L = (1,2)-SWAP(k, L, f_Inner)
10:         P = P ∪ {L}
11:         L = PERTURB(L, Uniform Random)
12:                                                 ▷ Phase 2: Penalty
13:     for 50 iterations do
14:         L = (1,2)-SWAP(k, L, f_Inner)
15:         P = P ∪ {L}
16:         L = PERTURB(L, Penalty)
17:                                                 ▷ Phase 3: Degree-Based
18:     for 100 iterations do
19:         L = (1,2)-SWAP(k, L, f_Inner)
20:         P = P ∪ {L}
21:         L = PERTURB(L, Degree-Based)
22:     if P is unchanged then
23:         k = k + 2
24: return P
```

(1) Uniform random selection chooses a vertex from C using a uniform distribution, so that each vertex not yet in the solution has an equal chance of being selected.
(2) Penalty selection acts by determining the frequency of solutions in P containing each vertex $v \in C$. A vertex is then selected using a Gaussian distribution on those frequencies, which maintains a random element in the selection, but favors those vertices that have not been present in many solutions. This can help jump out of local maxima during the search.
(3) Degree-based selection chooses a vertex $v \in C$ of largest degree. That is, $\deg(v) \geq \deg(w)$ for all $w \in C$.

After finishing the three phases, if no new solutions have been added to P, k is increased by 2. The entire algorithm is run until either a target count *maxSolns* of solutions is reached or a target count *maxIters* of iterations are completed. Instead of returning the best solution found as evaluated by f_{Inner}, all solutions $L \in P$ are returned, so they can be evaluated by $f_{\text{Outer}}(L)$, similar to clique enumeration. The values *maxSolns* and *maxIters*, and the functions $f_{\text{Inner}}(L)$ and $f_{\text{Outer}}(L)$, are taken as parameters to the algorithm, and can differ between applications. *maxIters* allows us to limit the execution time of the algorithm in cases where the number of available unique solutions is less than the target *maxSolns*. The heuristic evaluation function f_{Inner} should be representative of high-quality solutions, and be efficient to compute, since it directs the algorithm to explore or abandon entire branches of solutions, and must not throttle the computational efficiency of the search. Once the search completes, f_{Outer} is used to choose the optimal solution from all potential solutions, and may be less computationally efficient. Here, it is beneficial to have fewer potential solutions to evaluate in order to maintain speed while not sacrificing quality. We denote running PLISkE with *maxSolns* and *maxIters* by PLISkE(*maxIters*, *maxSolns*).

The complexity of PLISkE is defined by *maxIters*, *maxSolns*, $|P|$ and the time complexity of f_{Inner} and f_{Outer}. One iteration of

PLISkE takes $O(T(f_{\text{Inner}}) \cdot n^3 + n^5)$ time, since performing a $(1,2)$-swap dominates the running time; thus, the partial enumeration takes $O(\min\{maxIters, maxSolns\} \cdot (T(f_{\text{Inner}}) \cdot n^3 + n^5))$ time. In our experiments in the next section, we choose $maxIters \leq 500$, and therefore the running time is $O(T(f_{\text{Inner}}) \cdot n^3 + n^5)$.

The time complexity of finding the best solution from PLISkE's enumerated results is $O(T(f_{\text{Outer}}) \cdot |P|)$. Note that since a subgraph is added to P at most once in each phase of PLISkE, we have $|P| \leq maxIters \cdot (50 + 50 + 100)$ and therefore the running time of this step is $O(T(f_{\text{Outer}}))$. The entire algorithm therefore takes $O(T(f_{\text{Inner}}) \cdot n^3 + n^5 + T(f_{\text{Outer}}))$ time. Thus, unless $T(f_{\text{Inner}})$ or $T(f_{\text{Outer}})$ is c^n for some $c > 1$, which is not the case in our specific applications, PLISkE is faster than clique enumeration and k-plex enumeration.

4 APPLICATIONS OF PARTIAL k-PLEX ENUMERATION

To show the feasibility of partial k-plex enumeration, we run PLISkE on two applications that use graph frameworks. We begin with feature filtering from machine learning, using the same datasets as Schroeder et al. [15], which we then extend to graph instances of varying sizes and densities. We then show the efficacy of PLISkE on a novel application in the geosciences.

4.1 Feature Filters for Machine Learning

ML-based classification models are often trained using large, feature-rich datasets that may contain redundant or irrelevant features. These features often cause over-fitting, affect computational performance at training time, and lower the precision of the trained model [17]. These issues can be mitigated by selecting a subset of the available features. Schroeder et al. [15] introduced a framework to model features from a dataset and their relationships with one another as an undirected graph with vertex and edge weights. The edge weights are given by the absolute value of the Pearson Correlation ($w_E(v_i v_j) = p_{v_i v_j} \in [0,1]$) for the data of the two features, and each vertex v is weighted by the distance from the Kolmogorov-Smirnov distribution ($w(v) = k_v \in [0,1]$). A high edge weight indicates redundant features, and a low vertex weight indicates irrelevant features. The worst 10% of edges are removed, and all vertices with $k \leq 0.1$ are removed. They then enumerate all maximal cliques using a custom implementation of the worst-case optimal algorithm by Tomita et al. [18] and return a maximum clique $K = (V', E')$ that maximizes $f_{FF}(K) = 1 - \sum_{v_i v_j \in E'} \frac{w_E(v_i v_j)}{|E'|} + \sum_{v \in V'} \frac{w(v)}{|V'|}$. Due to computational constraints imposed by the original implementation by Schroeder et al. [15] only the 19 highest scoring vertices remain in the created graph. After correcting the implementation we were able to remove the 19 vertex constraint. Performance was evaluated by two criteria: feature count in the chosen subset (size), and fraction of points correctly classified by a ML algorithm (accuracy). We apply PLISkE using $f_{\text{Inner}} = f_{\text{Outer}} = f_{FF}$. All experiments were run on Intel(R) Xeon(R) E5-2620 v4 @ 2.10GHz 16 core CPU, and 1.5 TB of RAM, using Java 14.0.1.

Schroeder et al. [15] evaluate their method on 16 dataset/ML algorithm combinations formed by matching the Glass, Wine, Shuttle, and Ionosphere datasets from UCI [6], with machine learning models for Naive Bayes (NB), k-nearest neighbors (KNN), support vector machine (SVM), and Logistic Regression (LR) using the Weka library [8]. We introduce 4 more combinations by lifting the 19-vertex limit on Ionosphere (denoted Ionosphere+)—the only dataset with greater than 19 features.

We first ran experiments to find a suitable combination of hyperparameters, by varying $maxIters$ and $maxSolns$ on a subset of the dataset/ML combinations. We chose $maxSolns$ to be a function of n, to allow the number of solutions to scale with instance size. We considered both a linear and a quadratic number of candidate solutions, due to their computational feasibility. We did not move to cubic due to the increase in computational expense and promising results from the quadratic measurements. Values for $maxIters$ 'safeguard' against instances where $maxSolns$ candidate solutions cannot be found. As such these values must be large enough to allow most instances to find enough solutions, while not allowing very hard instances to take too much time. We felt that 100 and 500 were both reasonable for this task. The best results are achieved with PLISkE$(100, 0.5n^2)$, which achieves a balance of solution size, accuracy, and low runtime, and PLISkE$(500, 2n^2)$, which achieves a similar balance, but with slightly higher accuracy and runtime. We compare these two variants of PLISkE, and 2-plex and 3-plex enumeration against the original clique enumeration technique on all 20 dataset/ML combinations. The full results are omitted for brevity, so we include summary plots here. We ran PLISkE with 10 random seeds for each test, and present average values over these 10 runs.

The average relative differences of accuracy, size, and time between the original clique method and the four other methods are given in Table 1. Relative differences are used to provide a clear comparison between all methods within a metric, and between the various metrics. By removing units such as seconds and subgraph size we are able to more clearly see where improvements in one metric outweigh worse performance in another. All new methods have lower average accuracy than the clique enumeration method by less than 1%. However, both PLISkE methods find solutions that are at least 13% smaller.

Figures 1a-1c and Figures 1e–1g give *performance plots*, which summarize the performance of each method compared to the performance of the best method on each instance for a given metric. The x- and y-axes of each plot are ratios between 0 and 1. For each method, a point (x, y) on the plot indicates that the ratio y of its solutions are within the factor x of the best solution computed by any method, according to the metric. The x value at which a curve hits $y = 1$ is the largest factor of any solution from the best solution, while the y value at $x = 1$ indicates the ratio of best solutions that are found by a method. For these plots, faster growing curves (and those reaching $y = 1$ at the leftmost x value) indicate that a high ratio of solutions are close to the best solution.

Table 1 shows that both PLISkE$(100, 0.5n^2)$ and PLISkE$(500, 2n^2)$ find smaller (better) solutions than clique enumeration, by 23% and 13% on average respectively, while 2- and 3-plex enumeration both find larger (worse) solutions. As Figure 1a illustrates, PLISkE$(100, 0.5n^2)$ has the minimum solution size on all instances, and no other method finds a solution of minimum size. On all instances, the minimum solution size is at least a factor of 0.87 of PLISkE$(500, 2n^2)$'s solution. In other words, PLISkE$(500, 2n^2)$ is at most a factor of $1/0.87 = 1.14$ (14%) above the minimum solution size. The remaining methods have consistently larger solution sizes. The PLISkE$(100, 0.5n^2)$ variation also finds solutions fastest on average, which is shown in Table 1 and indicated in Figure 1b. Figure 1c shows that all methods have very similar accuracy, with the worst of all the runs being only 7% worse than the best. Additionally, both PLISkE variants have better worst cases than

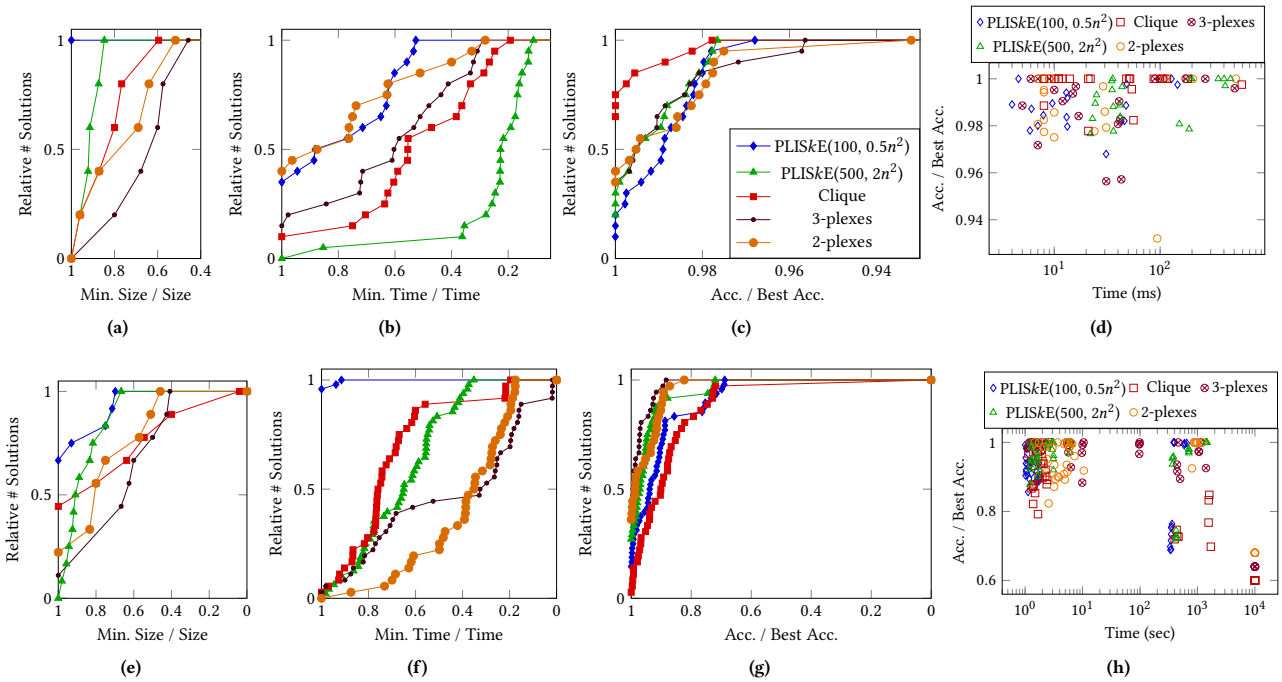

Figure 1: Results for feature filtering using clique and k-plex enumeration, and PLISkE variants, including performance plots for (a) solution size, (b) time, and (c) accuracy. (d) shows relative accuracy versus actual time for all methods. (a)-(d) are for glass, wine, shuttle, and ionosphere graphs. (e)-(h) are the same plots for all graphs created using the Dexter dataset. Note that k-plex enumeration was executed in C++, while clique enumeration and PLISkE were executed in Java. Time values plotted at 10 000s were terminated after 200 hours with no solution found.

the k-plex enumeration methods, with PLISkE(500, $2n^2$) performing best after the clique method. The implementation of the full k-plex enumeration algorithm (not PLISkE) was in C++, which is why those are faster than the clique enumeration (which is 1-plex enumeration). PLISkE(500, $2n^2$) reports the worst relative difference in time in Table 1. Figure 1d displays relative accuracy versus time for all instances, and demonstrates the time advantage given by using PLISkE(100, $0.5n^2$) by the grouping of points in the top left corner of the plot, which is the optimal position. This plot also shows that the best solution is most often found by clique enumeration, and occasionally by other methods.

The improvements in size by PLISkE are far greater than the losses in accuracy, and indicate the feasibility of using partial enumeration to solve problems with graph formulations. These results indicate that on these very small, contrived instances the clique is often the best solution in terms of the accuracy metric, which is a critical metric in ML applications such as this. However, we also clearly see an indication that the k-plex, and the various enumeration methods, are promising in terms of accuracy. Specifically in Figure 1c, PLISkE(500, $2n^2$) reaches $y = 1$ at the same point as the clique enumeration method, and PLISkE(100, $0.5n^2$) is close behind. PLISkE also outperforms all three full enumeration methods in several instances. PLISkE(100, $0.5n^2$) is best in terms of time, and of both PLISkE methods are best in terms of size. Thus the potential of the k-plex structure alongside the local search methods can not be overlooked. We now continue to explore the efficacy of these methods by observing their performance on problem instances with larger graphs.

Table 1: Average relative difference (R.D.) $\frac{2(x-y)}{(x+y)}$ over the glass, wine, shuttle, ionosphere, and ionosphere+ datasets between the results from the original clique method, 2- and 3-plex enumeration, and PLISkE averaged over 10 runs with random seeds. Values where the clique method is outperformed are in bold.

Method	R.D. Acc	R.D. Size	R.D. Time
PLISkE(100, $0.5n^2$)	−0.009	**−0.234**	**−0.449**
PLISkE(500, $2n^2$)	−0.006	**−0.133**	0.713
2-plexes	−0.009	0.093	**−0.402**
3-plexes	−0.008	0.253	**−0.188**

4.2 Scaling to Larger Instances

In order to demonstrate the effectiveness of PLISkE on a variety of larger graphs with varying densities we chose the Dexter dataset from the UCI repository that contains 2 600 entries each with 20 000 features [6]. We altered the graph creation framework by changing the threshold values for removing vertices and edges. We used the original edge threshold of removing the worst 10%, and added thresholds of $\text{avg}(\{w_E(e) \mid e \in E\})/x$, for $x \in \{2, 4, 8\}$, creating more sparse graphs. We removed $v \in V$ where $w(v) < \{0.005, 0.05, 0.1\}$, creating larger graphs. All possible combinations of these thresholds yielded 12 graphs with varying sizes and densities. Those graphs were sized as follows, given as (n, m): (4 280, 8 241 354), (4 280, 2 792 156), (4 280, 67 670), (4 280, 24 766), (165, 12 177), (165, 3 843), (165, 1 847), (165, 966), (52, 1 193), (52, 271), (52, 130), and (52, 70). We then compared PLISkE to clique and 2- and 3-plex enumeration.

Table 2: Average relative difference (R.D.) $\frac{2(x-y)}{(x+y)}$ on the graphs created from the Dexter dataset between the results from the original clique method, 2- and 3-plex enumeration, and PLISkE averaged over 10 runs with random seeds. Values where the clique method is outperformed are in bold.

Method	R.D. Acc	R.D. Size	R.D. Time
PLISkE(100, $0.5n^2$)	−0.006	**−0.360**	**−0.384**
PLISkE(500, $2n^2$)	**0.053**	**−0.291**	**−0.065**
2-plexes	**0.082**	**−0.118**	0.579
3-plexes	**0.101**	0.063	0.549

Table 2 compares the results of clique, 2- and 3-plex enumeration on these larger graphs with the results of PLISkE. Figures 1e–1g are performance plots analogous to those in Figures 1a–1c and display that data for the solutions of the Dexter graphs. Both PLISkE variants and 2-plex enumeration show improvement in solution size on average over clique enumeration. PLISkE(100, $0.5n^2$) finds the largest improvement, with a decrease in size of 36%, while PLISkE(500, $2n^2$) and 2-plex enumeration are better by 29% and 11% on average. Both PLISkE(100, $0.5n^2$) and PLISkE(500, $2n^2$) have shorter runtimes on average than clique enumeration by 38% and 6.5%, respectively. 2- and 3-plex enumeration ran slower, both with an average increase in runtime of over 50%. This is especially significant considering that clique enumeration was implemented in Java, while k-plexes were enumerated in compiled C++, before running f_{Outer} in Java. Figure 1e also demonstrates the strength of using k-plexes as solutions. The solutions found by any of the full enumeration methods were similar in size, with the 2-plexes being the best of the three in the worst case. However, the solutions by PLISkE(100, $0.5n^2$) were generally the smallest (best), and the solutions with the worst relative size from PLISkE(500, $2n^2$) were closer to the best than the worst found by each of the enumeration methods. Figure 1g demonstrates that as datasets grow in size the solutions are more likely to be k-plexes with $k > 1$. This also shows that a poor problem formulation can be overcome by k-plexes with $k > 1$.

In Table 2 we see that all methods except PLISkE(100, $0.5n^2$) find greater accuracy on average than clique enumeration. 3-plex enumeration has the greatest improvement over clique enumeration, by 10%, followed by 2-plex enumeration at 8% and PLISkE(500, $2n^2$) at 5%. PLISkE(100, $0.5n^2$) only shows a 0.5% decrease in accuracy on average. Figure 1h compares the relative accuracy of each solution to the true time spent finding it, with the upper left corner being optimal. The 3 points in the lower right corner represent 3 graphs that the enumeration methods could not fully enumerate in under 200 hours, even when running a worst-case optimal clique enumeration method from Tomita et al. [18]. The cluster of points on the top left of the plot has a group of points from PLISkE(100, $0.5n^2$) on the left, indicating the faster speed with still-promising accuracy. The points from PLISkE(500, $2n^2$) are also clustered more tightly to the top left of the plot, showing that they are on average better than the enumeration methods. The other points on the right side of the plot are from the more challenging instances, and show the speed advantage and comparable accuracy of the PLISkE variants relative to enumeration. These results show that using full k-plex enumeration gives a large number of potential solutions, which significantly increases the runtime of the algorithm.

The results obtained with k-plex enumeration on these larger graphs indicate the strength of using k-plexes when the graph formulation or data is flawed or the graph created is large and sparse. Further, the significant decrease in runtime when using PLISkE, combined with the comparable or better solutions in terms of size and accuracy, indicates that PLISkE strikes a beneficial balance between the metrics observed over either clique or k-plex enumeration. The only method that on average outperforms the clique enumeration in terms of accuracy, solution size, and speed is PLISkE(500, $2n^2$). This further supports using both the k-plex structure and partial enumeration.

4.3 Further Applications of PLISkE

We used the Java implementation of PLISkE for a new application in the geosciences based on work by Conrey et al. [4]. This application is used to further illustrate the value of using k-plexes and partial enumeration.

Determining the elemental composition of geological samples is either expensive or reliant on machines with low precision. Conrey et al. [4] propose a method to improve accuracy of inexpensive methods by leveraging correlations between elements to manually choose elements that are the best indicators of the presence of other elements. However, the time needed by an expert to manually choose these sets of elements is over eight hours per dataset, even when using specially-designed software [4].

We introduce a novel graph formulation to compute these element sets with PLISkE. The vertices in the graph $G = (V, E)$ represent the elements, and the edges represent the correlations between the elements. Weights are assigned to the edges by $w_E(v_s v_t) = corr(s, t)$, where $corr(s, t)$ is the correlation between the two elements s and t at the vertices $v_s, v_t \in V$, respectively. Edges are removed from E if their weight falls below a threshold calculated by taking the average of the values at the 80th percentile of $\{corr(s, t) \mid \text{for any elements } s, t\}$. For each target element t we then create a unique graph, $G_t = (V_t, E_t)$, where $V_t = V \setminus \{v_t\}$ and $w(v_s) = corr(s, t)$ for $v_s \in V_t$.

According to Rick Conrey, a domain expert, the most important features of an ideal element set and the final correlation equation are, in this order, larger size of the element set, higher r^2 correlation, and a y-intercept close to 0. For a given subgraph $K = (V_K, E_K)$, we choose $f_{\text{Inner}}(K) = \frac{1}{\log(|V_K|)} \sum_{v \in V_K} w(v)$, for its low computational cost and since it favors larger solutions, even if the average vertex weight is slightly lower. For the target element t we define $f_{\text{Outer}}(K) = (r_2(t, V_K) - 0.005x_1(t, V_K))1.5^{|V_K|}$, where $r_2(t, V_K)$ denotes the r^2 value of the correlation and x_1 is the y-intercept for the model for t, calculated using the elements in V_K. To evaluate the solution quality, we consider size and r^2.

We ran our experiments on a dataset of 54 elements that we call GT1. GT1 contains reference values for 41 elements selected manually by Conrey, and is the outcome of a standard attempt to solve the entire dataset, parts of which are too difficult to solve manually (the 13 remaining elements are unsolved). We compare PLISkE and maximal clique enumeration using Tomita et al. [18]. All experiments were run on macOS 10.15.6, Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz 8 core CPU, and 32 GB RAM, using Java 14.0.1. The results of parameter tuning on six of the GT1 instances are omitted for brevity. As a result, we consider two variants, PLISkE(100, $0.5n^2$), which achieves a balance of solution size, r^2, and low runtime, and PLISkE(500, $4n^2$), which achieves the largest size and high r^2 values at the cost of greater runtime.

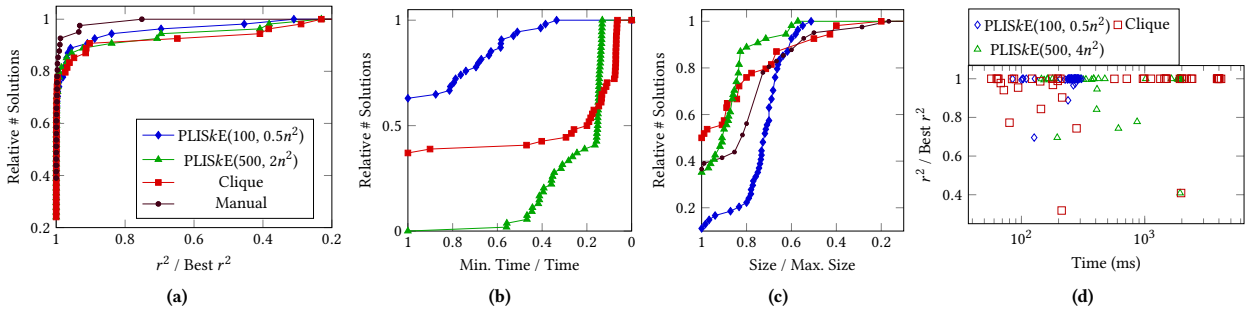

Figure 2: Comparing clique enumeration, manual solving (no time value reported), and PLISkE variants for the geosciences application on the GT1 dataset. Performance plots for (a) r^2, (b) time, and (c) solution size. (d) Relative r^2 versus actual time for all methods.

Figure 2 compares the results of manually solving, clique enumeration, PLISkE(100, 0.5n^2), and PLISkE(500, 4n^2). Figure 2a shows that the clique method finds the worst relative r^2 on several instances. Figure 2c shows that both PLISkE variants find solutions with size closer to the best than the manual or clique methods in the worst case for all methods. Figure 2b shows that PLISkE(100, 0.5n^2) is faster than the other methods in most cases, and in the worst case. Additionally, PLISkE(500, 4n^2) is closer to the best case for runtime than clique enumeration on almost 50% of instances. Figure 2d also demonstrates that the worst case runtime for clique enumeration is much greater than for either PLISkE variation. It also shows that most PLISkE(100, 0.5n^2) runs are between 80ms and 400ms, while the clique instances range from under 30ms to 4000ms. Most solutions have relative $r^2 > 0.9$, with the worst cases for clique, PLISkE(500, 4n^2), and PLISkE(100, 0.5n^2) at 35%, 40%, and 70%, respectively. By considering the solutions that were not solved manually in GT1 we see that PLISkE provides overall better results on hard instances for this application. Whereas manually selecting element sets for these 54 instances requires more than 8 hours of manual work for a domain expert, both PLISkE variants finish in under a minute.

Overall, these results further indicate the efficacy of the k-plex structure as a solution in imperfect graph formulations, especially when paired with a heuristic-directed local search that lends advantages in both runtime and solution quality.

5 CONCLUSION

We proposed a technique to partially enumerate k-plexes using local search. Our proof-of-concept algorithm, PLISkE, finds solutions of higher quality than with clique enumeration, and more quickly than with k-plex enumeration, in experiments with feature filtering and an application in the geosciences. Important future work includes optimizing the implementation, tuning parameters, and exploring different local search strategies.

REFERENCES

[1] Diogo V Andrade, Mauricio G C Resende, and Renato F Werneck. 2012. Fast local search for the maximum independent set problem. *J. Heuristics* 18 (2012), 525–547. https://doi.org/10.1007/s10732-012-9196-4

[2] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223. https://doi.org/10.1137/0214017

[3] F. A. Rezaur Rahman Chowdhury, Chao Ma, Md Rakibul Islam, Mohammad Hossein Namaki, Mohammad Omar Faruk, and Janardhan Rao Doppa. 2017. Select-and-evaluate: A learning framework for large-scale knowledge graph search. In *Proc. the Ninth ACML*, Vol. 77. PMLR, 129–144. http://proceedings.mlr.press/v77/chowdhury17a/chowdhury17a.pdf

[4] Richard M. Conrey, David G. Bailey, Jared W. Singer, Laureen Wagoner, Benjamin Parfitt, John Hay, and Oliver Keh. 2019. Optimization of internal standards in LA-ICPMS analysis of geologic samples using lithium borate fused glass. *Geological Society of America Abstracts with Programs* 54, 1 (2019). https://doi.org/10.1130/abs/2019NE-328672

[5] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. 2020. Sublinear-space and bounded-delay algorithms for maximal clique enumeration in graphs. *Algorithmica* 82, 6 (2020), 1547–1573. https://doi.org/10.1007/s00453-019-00656-8

[6] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository. http://archive.ics.uci.edu/ml

[7] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *ACM J. Exp. Algorithmics* 18, Article 3.1 (2013), 21 pages. https://doi.org/10.1145/2543629

[8] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software. *ACM SIGKDD Explorations Newsletter* 11, 1 (2009), 10–18. https://doi.org/10.1145/1656274.1656278

[9] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of Metaheuristics*. Springer, 320–353. https://doi.org/10.1007/0-306-48056-5_11

[10] Kazuhisa Makino and Takeaki Uno. 2004. New algorithms for enumerating all maximal cliques. In *Proc. SWAT 2004 (LNCS, Vol. 3111)*, Torben Hagerup and Jyrki Katajainen (Eds.). Springer, 260–272. https://doi.org/10.1007/978-3-540-27810-8_23

[11] J W Moon and L Moser. 1965. On cliques in graphs. *Israel J. Mathematics* 3, 1 (1965), 23–28. https://doi.org/10.1007/BF02760024

[12] Feiping Nie, Wei Zhu, and Xuelong Li. 2016. Unsupervised feature selection with structured graph optimization. In *Proc. the 30th AAAI*. AAAI, 1302–1308. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12180

[13] Wayne Pullan. 2006. Phased local search for the maximum clique problem. *J. Combinatorial Optimization* 12, 3 (2006), 303–323. https://doi.org/10.1007/s10878-006-9635-y

[14] Wayne Pullan. 2009. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization* 6, 2 (2009), 214–219. https://doi.org/10.1016/j.disopt.2008.12.001

[15] Daniel Thilo Schroeder, Kevin Styp-Rekowski, Florian Schmidt, Alexander Acker, and Odej Kao. 2019. Graph-based Feature Selection Filter Utilizing Maximal Cliques. In *Proc. SNAMS 2019*. IEEE, 297–302. https://doi.org/10.1109/SNAMS.2019.8931841

[16] Stephen B Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *J. Mathematical Sociology* 6, 1 (1978), 139–154. https://doi.org/10.1080/0022250X.1978.9989883

[17] Jiliang Tang, Salem Alelyani, and Huan Liu. 2014. Feature selection for classification: A review. In *Data Classification*. CRC Press, 37–64. https://doi.org/10.1201/b17320

[18] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science* 363, 1 (2006), 28–42. https://doi.org/10.1016/j.tcs.2006.06.015

[19] Chenliang Xu, Caiming Xiong, and Jason J. Corso. 2012. Streaming hierarchical video segmentation. In *Proc. the 12th ECCV, Part VI*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid (Eds.). Vol. 7577 LNCS. Springer, 626–639. https://doi.org/10.1007/978-3-642-33783-3_45

[20] Dong Zhang, Omar Javed, and Mubarak Shah. 2014. Video object co-segmentation by regulated maximum weight cliques. In *Proc. the 13th ECCV, Part VII*. Vol. 8695 LN. Springer, 551–566. https://doi.org/10.1007/978-3-319-10584-0_36

[21] Yi Zhou, Jingwei Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating maximal k-plexes with worst-case time guarantee. In *Proc. the 34th AAAI*. 2442–2449. https://doi.org/10.1609/aaai.v34i03.5625