An Improved Parameterized Algorithm for a Generalized Matching Problem*

Jianxin Wang, Dan Ning, Qilong Feng, and Jianer Chen

School of Information Science and Engineering, Central South University
jxwang@mail.csu.edu.cn

Abstract. We study the parameterized complexity of a generalized matching problem, the P_2-packing problem. The problem is NP-hard and has been studied by a number of researchers. In this paper, we provide further study of the structures of the P_2-packing problem, and propose a new kernelization algorithm that produces a kernel of size $7k$ for the problem, improving the previous best kernel size $15k$. The new kernelization leads to an improved algorithm for the problem with running time $O^*(2^{4.142k})$, improving the previous best algorithm of time $O^*(2^{5.301k})$.

1 Introduction

Packing problem has formed an important class of NP-hard problems. In particular, as one of the graph packing problem, the H-packing problem has gained more attention, which arises in applications such as scheduling, wireless sensor tracking, wiring-board design and code optimization, etc. The problem is defined as follows [1].

Definition 1. Given a graph $G = (V, E)$ and a fixed graph H. An H-packing of G is a set of vertex disjoint subgraphs of G, each of which is isomorphic to H.

From the optimization point of view, the problem of MAXIMUN H-packing is to find the maximum number of vertex disjoint copies of H in G. If the H is the complete graph K_2, the MAXIMUN H-packing becomes the familiar maximum matching problem in bipartite graph, which can be solved in polynomial time. When the graph H is a connected graph with at least three vertices, D. G. Kirkpatrick and P. Hell [2] gave that the problem is NP-complete. From the approximation point of view, V. Kann [3] proved that the MAXIMUN H-packing problem is MAX-SNP-complete. C. Hurkens and A. Schrijver [4] presented an approximation algorithm with ratio $|V_H|/2 + \varepsilon$ for any $\varepsilon > 0$.

* This work is in part supported by the National Natural Science Foundation of China under Grant No. 60773111 and No. 60433020, Provincial Natural Science Foundation of Hunan (06JJ10009), the Program for New Century Excellent Talents in University No. NCET-05-0683 and the Program for Changjiang Scholars and Innovative Research Team in University No. IRT0661.

M. Agrawal et al. (Eds.): TAMC 2008, LNCS 4978, pp. 212–222, 2008.

Recently, parameterized complexity theory has been used to design efficient algorithms for H-packing problem. M. Fellows et. al. [5] proposed a parameterized algorithm with time complexity of $O(2^{O(|H|k \log k + k|H| \log |H|)})$ for any arbitrary graph H. For the edge disjoint triangle packing problem, L. Mathieson, E. Prieto and P. Shaw [6] proved that the problem has a $4k$ kernel and gave a parameterized algorithm of running time $O(2^{\frac{9k}{2} \log k + \frac{9k}{2}})$ based on the kernel.

When H belongs to the restricted family of graphs $K_{1,s}$, a star with s leaves, we can get the $K_{1,s}$-packing problem, which is defined as follows:

Definition 2. Parameterized $K_{1,s}$-PACKING(k-$K_{1,s}$-PACKING): Given a graph $G = (V, E)$ and a positive integer k, whether there are at least k vertex disjoint $K_{1,s}$ in G?

M. Fellows, E. Prieto and C. Sloper [7] gave that the parameterized $K_{1,s}$-packing problem is fixed-parameter tractable and got a $O(k^3)$ kernel. M. Fellows [7] et.al. also studied the P_2-packing problem, where P_2 is a path of three vertices (one center vertex and two endpoints) and two edges, which is defined as follows:

Definition 3. Parameterized P_2-PACKING (k-P_2-PACKING): Given a graph $G = (V, E)$ and a positive integer k, whether there are at least k vertex disjoint P_2 in G?

In [7], for the P_2-packing problem, M. Fellows et.al. gave a kernel of size at most $15k$ and proposed an algorithm with time complexity $O^*(2^{5.301k})$.

In this paper, we mainly focus on the kernelization of the k-P_2-packing problem and give a kernel of size at most $7k$. Based on the kernel, we present a parameterized algorithm with time complexity $O^*(2^{4.142k})$, which greatly improves the current best result $O^*(2^{5.301k})$.

This paper is organized as follows. In section 2, we introduce some related definitions and lemmas. In section 3, we present all the steps of the kernelization algorithm, and prove that the k-P_2-packing problem has a size of $7k$ kernel. In section 4, we give the general algorithm solving the k-P_2-packing. In section 5, we draw some final conclusions.

2 Related Definitions and Lemmas

We first give some concepts and terminology about graph [8].

Assume $G = (V, E)$ denotes a simple, undirected, connected graph, where $|V| = n$. The neighbors of a vertex v are denoted as $N(v)$. The induced subgraph of $S \subseteq V$ is denoted $G[S]$. For an arbitrary subgraph H of G, let $N(H)$ denote the vertices that are not in H but connect with at least one vertex in H. We use the simpler $G \backslash v$ to denote $G[V \backslash v]$ for a vertex v and $G \backslash e$ to denote $G = (V, E \backslash e)$ for an edge e. Likewise, $G \backslash V'$ denotes $G[V \backslash V']$ and $G \backslash E'$ denotes $G = (V, E \backslash E')$ where V' is a set of vertices and E' is a set of edges.

For the convenience of description, we firstly introduce the definitions of 'double crown' decomposition and 'fat crown' decomposition [7].

Definition 4. A double crown decomposition (H, C, R) in a graph $G = (V, E)$ is a partitioning of the vertices of the graph into three sets H, C and R that have the following properties:

(1) H (the head) is a separator in G such that there are no edges in G between vertices belonging to C and vertices belonging to R.

(2) $C = C_u \cup C_m \cup C_{m2}$ (the crown) is an independent set in G.

(3) $|C_m| = |H|$, $|C_{m2}| = |H|$ and there is a perfect matching between C_m and H, and a perfect matching between C_{m2} and H.

Definition 5. A fat crown decomposition (H, C, R) in a graph $G = (V, E)$ is a partitioning of the vertices of the graph into three sets H, C and R that have the following properties:

(1) H (the head) is a separator in G such that there are no edges in G between vertices belonging to C and vertices belonging to R.

(2) $G[C]$ is a forest where each component is isomorphic to K_2.

(3) $|C| \geq |H|$, and there is a perfect matching M between H and a subset of cardinality $|H|$ in C, where one endpoint of each edge in M is in H, and the other is the endpoint of K_2 in C.

We introduce the following lemmas [7] about the 'double crown' decomposition and 'fat crown' decomposition that will be used in our algorithm.

Lemma 1. *A graph $G = (V, E)$ that admits a 'double crown'-decomposition (H, C, R) has a k-P_2-packing if and only if $G \backslash (H \cup C)$ has a $(k - |H|)$-P_2-packing.*

Lemma 2. *A graph $G = (V, E)$ that admits a 'fat crown'-decomposition (H, C, R) has a k-P_2-packing if and only if $G \backslash (H \cup C)$ has a $(k - |H|)$-P_2-packing.*

Lemma 3. *A graph G with an independent set I, where$|I| \geq 2|N(I)|$, has a double crown decomposition (H, C, R), $H \subseteq N(I)$, which can be constructed in linear time.*

Lemma 4. *A graph G with a collection J of independent K_2s, where $|J| \geq |N(J)|$, has a fat crown decomposition (H, C, R), $H \subseteq N(J)$, which can be constructed in linear time.*

3 Kernelization Algorithm for the k-P_2-Packing Problem

In this section we propose a kernelizaiton algorithm that can get a kernel of size at most $7k$ for the parameterized version of P_2-packing problem.

Assume W denotes a maximal P_2-packing and the vertices in W are denoted by $V(W)$. Let W be $\{L_1, ..., L_t\}$, $t \leq k - 1$, where each of $L_i (1 \leq i \leq t)$ is a subgraph in G that is isomorphic to P_2. Let L_i be (e_1, c, e_2), $1 \leq i \leq t$, where e_1 and e_2 are two endpoints of L_i, and c is the center vertex of L_i. Therefore, each connected component of the graph induced by $Q = V \backslash V(W)$ is either a single vertex or a single edge [7]. Let Q_0 be the set of all vertices such that each vertex in Q_0 makes a connected component of the graph induced by Q, and each vertex in Q_0 will be called a Q_0-vertex. Let Q_1 be the set of all edges such that each edge in Q_1 makes a connected component of the graph induced by Q. Each edge in Q_1 will be called a Q_1-edge and each vertex in Q_1 will be called a Q_1-vertex.

3.1 RPLW Algorithm

Based on the kernelization algorithm given in [7], the kernelization process we propose is to apply the algorithm RPLW repeatedly. By using the kernelization algorithm in [7], we can get a graph G, which consists of a maximal packing W and $Q = V \backslash V(W)$. The algorithm RPLW is to further reduce the vertices in W and Q to get a better kernel, whose general idea is given in the following:

Algorithm RPLW deals with the Q_0-vertices and Q_1-edges in Q. When the size of W is not changed, the algorithm aims at reducing the number of Q_0-vertices in Q. When the number of Q_1-edges in Q is reduced, the size of W becomes larger (the number of disjoint P_2 in W is increased) and the algorithm returns the larger W. Then we call the algorithm for the larger W. If the 'double crown' decomposition or the 'fat crown' decomposition is found, the parameter k becomes smaller and the algorithm returns the smaller parameter. Then we call the algorithm for the smaller parameter.

For the convenience of analyzing the RPLW algorithm, we first discuss the following two structures as shown in Fig.1 and Fig.2, which use solid circles and thick lines for vertices and edges in the maximal P_2-packing W, and use hollow circles and thin lines for vertices and edges not in W. (In particular, thin lines that connect two hollow circles are Q_1-edge.)

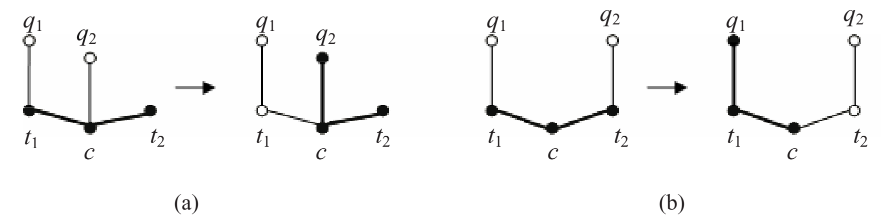

Fig. 1. Reduce the number of Q_0-vertex

The general idea of Fig.1 is that: In order to decrease the number of Q_0-vertices in Q, replace the L_i in W. The specific process is as follows.

In Fig.1(a), assume the P_2 is the L_i in W whose center vertex is c and two endpoints are t_1, t_2. Q_0-vertex q_2 is adjacent to c, and Q_0-vertex q_1 is adjacent to t_1. Vertices q_2, c and t_2 can form a new P_2. Let the new P_2 be L_i'. If L_i is replaced by L_i' in W, the number of Q_0-vertices in Q is just reduced by 2 (q_1 and t_1 form a Q_1-edge in Q).

In Fig.1(b), assume the P_2 is the L_i in W whose center vertex is c and two endpoints are t_1, t_2. Q_0-vertex q_2 is adjacent to t_2, and Q_0-vertex q_1 is adjacent to t_1. Vertices $q_1 t_1$ and c can form a new P_2. Let the new P_2 be L_i'. If L_i is replaced by L_i' in W, the number of Q_0-vertices in Q is just reduced by 2 (q_2 and t_2 form a Q_1-edge in Q).

The general idea of Fig.2 is that: In order to decrease the number of Q_1-edges in Q and increase the number of disjoint P_2 in W, replace L_i in W . The specific process is as follows.

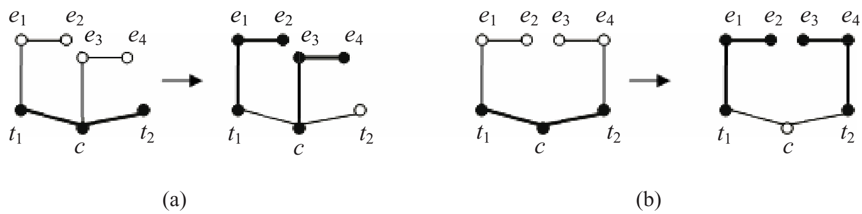

Fig. 2. Reduce the number of Q_1-edge

In Fig.2(a), assume the P_2 is the L_i in W whose center vertex is c and two endpoints are t_1, t_2. Q_1-edge (e_1, e_2) is adjacent to t_1, and Q_1-edge (e_3, e_4) is adjacent to c. Vertices e_1, e_2 and t_1 can form a new P_2, which can be denoted as L_i'. Vertices e_3, e_4 and c can also form a new P_2, which is denoted as L_i''. If L_i is replaced by L_i'' and L_i' in W, the number of Q_1-edges in Q is just reduced by 1 and the number of P_2 in W is increased by 1.

In Fig.2(b), assume the P_2 is the L_i in W whose center vertex is c and two endpoints are t_1, t_2. Q_1-edge (e_1, e_2) is adjacent to t_1, and Q_1-edge (e_3, e_4) is adjacent to t_2. Vertices e_1, e_2 and t_1 can form a new P_2 which is denoted as L_i'. Vertices e_3, e_4 and t_2 can also form a new P_2 which is denoted as L_i''. If L_i is replaced by L_i'' and L_i' in W , the number of Q_1-edges in Q is just reduced by 1and the number of P_2 in W are increased by 1.

Fig.1 and Fig.2 vividly illustrate how to replace a L_i in W to change the number of Q_0-vertices and Q_1-edges. According to Fig.1 and Fig.2, we can obtain the following rules.

Rule1. If a L_i in W has two vertices that each is adjacent to a different Q_0-vertex, then apply the processes described in Fig.1 to decrease the number of Q_0-vertices by 2 (and increase the number of Q_1-edges by 1).

Rule2. If a L_i in W has two vertices that each is adjacent to a different Q_1-edge, then apply the processes described in Fig.2 to decrease the number of Q_1-edges by 1(and increase the size of the maximal P_2-packing by 1).

The RPLW algorithm tries to reduce the number of Q_0-vertices and the number of Q_1-edges by applying Rule1 and Rule2 consecutively. Note that these rules cannot be applied forever. As shown in Fig.3, the while-loop in step1 of the algorithm tries to reduce the number of Q_0-vertices in Q. Because the number of vertices in input graph G is limited (at most $15k$ [7]) and each applications of Rule1 reduces the number of Q_0-vertices by 2, the number of consecutive applications of Rule1 is bonded by $7.5k$. During the applications of these rules, the resulting W may becomes non-maximal. In this cases, we simply first make W maximal again, using any proper greedy algorithm in step2 of the algorithm before we further apply the rules. Thus, the P_2 founded in Q can be put into W to make W larger. Assume the larger packing is W', then call the algorithm for W'.

During the process of the replacement of Q_1-edges in step3 of the algorithm, since each application of Rule2 increases the number of P_2 in W by 1, the

Algorithm RPLW
Input: G, W, k
Output: a maximal P_2-packing W and $|W'| > |W|$, or a smaller parameter k' and $|k'| < |k|$, or a reduced graph G'

1. **while** W is a maximal P_2-packing and a P_2 in W has two vertices that each is adjacent to two different Q_0-vertices **do**
   apply Rule1 to replace W by a packing of the same size with reduced Q_0-vertices;
2. **if** W is not maximal **then**
   use greedy algorithm to construct a larger P_2-packing W';
   return (G, W', k).
3. **if** two Q_1-edges are adjacent to two different vertices on a P_2 in W **then**
   apply Rule2 to obtain a larger P_2-packing W';
   return (G, W', k).
4. **if** $|Q_0| \geq 2|W|$ **then**
   construct a double crown decompositon (H, C, R), then $k' = k - |H|$;
   return (G, W, k').
5. **if** $|Q_1| \geq |W|$ **then**
   construct a fat crown decompositon (H, C, R), then $k' = k - |H|$;
   return (G, W, k').
6. Assume the reduced graph is G', return (G', W, k).

Fig. 3. RPLW algorithm

total number of applications of Rule2 is bounded by k. Step4 and step5 of the algorithm aim at finding ‘double crown’ and ‘fat crown’ in G induced by the replacement of Q_0-vertices and Q_1-edges. Once ‘double crown’ or ‘fat crown’ is found, the parameter k must be reduced ($k' = k - |H|$).

For completeness, we verify the algorithm’s correctness, and analyze its precise complexity.

Lemma 5. *Repeatedly calling the algorithm RPLW will either find a k-P_2-packing or reduce the size of G, and those can be done in $O(k^3)$.*

Proof. From step2 and step3 of the RPLW algorithm, it can be seen that the number of disjoint P_2 in W is increased by the replacement of Q_0-vertices and Q_1-edges. By calling the algorithm repeatedly, when the number of disjoint P_2 in W is k, a k-P_2-packing is found in graph G. Because of the replacements in step4 and step5 of the algorithm, ‘double crown’-decomposition or ‘fat crown’-decomposition will be found in G. Therefore, the parameter k is decreased and the number of disjoint P_2 needed to be found is also decreased. By calling the algorithm repeatedly, when the parameter k is reduced to 0, a k-P_2-packing is found in graph G. On the other hand, because the replacement in step1-3 of the algorithm limits the number of Q_0-vertices and Q_1-edges, and some vertices are removed by the ‘double crown’-decomposition or ‘fat crown’-decomposition in step4-5 of the algorithm, the size of G will be reduced. Thus, if a k-P_2-packing is not found in the algorithm, the algorithm returns a reduced G'.

At last, we analyze the time complexity of those whole process. Calling the RPLW algorithm repeatedly is to apply Rule1 and Rule2 consecutively, which can be finished in polynomial time. The number of consecutive applications of Rule1 is bonded by $7.5k$, and the total number of applications of Rule2 is bounded by k. When 'double crown'-decomposition or 'fat crown'-decomposition is applicable, the parameter k is reduced accordingly. The 'double crown'-decomposition or 'fat crown'-decomposition can be founded in $O(k^2)$ [7]. The algorithm must return when the number of disjoint P_2 in W is increased or the parameter k is reduced, and the algorithm is called again for the larger packing W' or the smaller parameter k'. If a k-P_2-packing is not found in the algorithm, the algorithm returns a reduced G'. Therefore, the algorithm is called at most $9.5k$ times. In consequence, the whole process can be computed in $O(k^3)$ time. □

Our kernelization process is to apply the RPLW algorithm repeatedly, i.e, to apply Rule1 and Rule2 repeatedly by starting with a maximal P_2-packing. The whole process can finish in polynomial time. The kernelization will either find a k-P_2-packing or reduce the size of G until Rule1 and Rule2 are not applicable. The reduced G can be considered as a kernel of the k-P_2-packing problem. Note that when Rule1 and Rule2 are not applicable, the maximal P_2-packing W (for each L_i in W) has the following properties:

Property 1. If more than one Q_0-vertices are adjacent to L_i, then all these Q_0-vertices must be adjacent to the same (and unique) vertex in L_i.

Property 2. If more than one vertex in L_i are adjacent to Q_0-vertices, then all these vertices in L_i must be adjacent to the same (and unique) Q_0-vertex.

Property 3. If more than one Q_1-edges are adjacent to L_i, then all these Q_1-edges must be adjacent to the same (and unique) vertex in L_i.

Property 4. If more than one vertex in L_i are adjacent to Q_1-edges, then all these vertices in L_i must be adjacent to the same (and unique) Q_1-edge.

3.2 A Smaller Kernel

In the following, we first analyze the number of Q_0-vertices and Q_1-edges in Q after the kernelization. Then we will present how the kernel of size at most $7k$ is obtained for k-P_2-packing problem.

We first analyze the number of Q_0-vertices in Q.

Theorem 1. *The number of Q_0-vertices is bounded by $2(k-1)$, that is, $|Q_0$-vertex$| \leq 2(k-1)$, or else we can find a double crown decomposition in polynomial time.*

Proof. When Rule1 and Rule2 are not applicable, let W be the maximal packing $W = L_1, \cdots, L_t$, $t \leq k-1$, which is a collection of disjoint P_2. We partition the disjoint P_2 in W into two groups: $\{L_1, \cdots, L_d\}$, $\{L_{d+1}, \cdots, L_t\}$, which satisfy the following property: for each L_i, $1 \leq i \leq d$, each Q_0-vertex adjacent to L_i can be adjacent to more than one vertex in L_i, and we denote these Q_0-vertex as Q_{0i} $(1 \leq i \leq d)$; for each L_j, $j > d$, each Q_0-vertex adjacent to L_j is at most adjacent to one vertex in L_j.

Consider the vertex set Q_0'-vertex=Q_0-vertex$-\{Q_{01}, \cdots, Q_{0d}\}$, and let $W' = \{v_1, \cdots, v_s\}$ be the set of vertices in $L_{d+1} \cup \cdots \cup L_t$ such that each vertex in W' has neighbor in Q_0-vertex. By the above partition property, each $L_j (j > d)$ has at most one vertex in W'. Thus, $s \leq t - d$. Moreover, by Property2, no vertex in Q_0'-vertex is adjacent to any L_i. Therefore, each vertex in Q_0'-vertex has all its neighbors in W', that is, $W' = N(Q_0'$-vertex).

Assume the total number of vertices in Q_0' -vertex is p. If $p > 2s$, there is a 'double crown'-decomposition in the input graph (note that the set of Q_0'-vertex is an independent set). We can call the RPLW algorithm again, which contradicts that Rule1 and Rule2 are not applicable. On the other hand, if $p \leq 2s$, the total number of Q_0-vertices in the graph is that: $|Q_0$-vertex$| = |Q_0'$-vertex$| + d = p + d \leq 2s + d \leq 2(s + d) \leq 2t \leq 2(k - 1)$. This completes the proof. □

In the following, we analyze the number of Q_1-vertices in Q.

Theorem 2. *The number of Q_1-vertices is bounded by $2(k - 1)$, that is, $|Q_1$-edge$| \leq k - 1$, or else we can find a double crown decomposition in polynomial time.*

Proof. When Rule1 and Rule2 are not applicable, let W be the maximal packing $W = L_1, \cdots, L_t$, $t \leq k - 1$, which is a collection of disjoint P_2. We partition the disjoint P_2 in W into two groups: $\{L_1, \cdots, L_d\}$, $\{L_{d+1}, \cdots, L_t\}$, which satisfy the following property: for each L_i, $1 \leq i \leq d$, each Q_1-edge adjacent to L_i can be adjacent to more than one vertex in L_i, and we denote these Q_1-edges as Q_{1i} $(1 \leq i \leq d)$; for each L_j, $j > d$, each Q_1-edge adjacent to L_j is at most adjacent to one vertex in L_j.

Consider the vertex set Q_1'-edge=Q_1-edge$-\{Q_{11}, \cdots, Q_{1d}\}$, and let $W' = \{v_1, \cdots, v_s\}$ be the set of vertices in $L_{d+1} \cup \cdots \cup L_t$ such that each vertex in W' has neighbors in Q_1-vertex. By the above partition property, each $L_j (j > d)$ has at most one vertex in W'. Thus, $s \leq t - d$. Moreover, by Property4, no vertex in Q_1'-edge is adjacent to any L_i, $1 \leq i \leq d$. Therefore, each vertex in Q_1'-edge has all its neighbors in W', that is, $W' = N(Q_1'$-edge).

Assume the total number of edges in Q_1'-edge is p. If $p > s$, there is a 'fat crown'-decomposition in the input graph (note that the set of Q_1'-edge is an independent set of K_2). We can call the RPLW algorithm again, which contradicts that Rule1 and Rule2 are not applicable. On the other hand, if $p \leq s$, the total number of Q_1-edges in the graph is that: $|Q_1$-edge$| = |Q_1'$-edge$| + d = p + d \leq s + d \leq t \leq k - 1$. Each Q_1-edge has two Q_1-vertices, therefore, the number of Q_1-vertices is bounded by $|Q_1$-vertex$| \leq 2(k - 1)$. This completes the proof. □

Based on theorem 1 and theorem 2, we can get the following theorem.

Theorem 3. *The k-P_2-packing problem has a kernel of size at most $7k - 7$.*

Proof. By applying the RPLW algorithm repeatedly until the two rules are not applicable. The vertices in G consist of the vertices in W and Q. Assume $V(G)$ denotes the vertices in G. The vertices in Q contains only Q_0-vertices and Q_1-vertices. By Theorem1, we can get that $|Q_0\text{-vertex}| \leq 2(k-1)$. By Theorem2, we can get that $|Q_1\text{-vertex}| \leq 2(k-1)$. Since $|V(W)| \leq 3(k-1)$, thus, $|V(G)| = |V(W)| + |Q_0| + |Q_1| \leq 3(k-1) + 2(k-1) + 2(k-1) = 7k - 7$. Therefore, the k-P_2-packing problem has a kernel of size at most $7k - 7$. □

4 The Improved Parameterized Algorithm

For the k-P_2-packing problem, we proposed an improved parameterized algorithm based on the $7k$ kernel. We first apply the kernelizaiton algorithm to obtain a kernel for the problem. Since each P_2 has a center vertex, in order to find k vertex disjoint P_2, we just need to find k center vertices in brute force manner on the $7k$ kernel. The specific algorithm is given in figure 4.

Algorithm KPPW
Input: $G = (V, E)$
Output: a k-P_2-packing in G, or can not find a k-P_2-packing in G

1. compute a maximal P_2-packing W with a greedy algorithm;
2. apply the RPLW(G, W, k) until rule1 and rule2 are not applicable;
3. **if** $|V(G)| > 7k$ **then**
   report "there exists a k-P_2-packing" and stop;
4. find all possible subsets C of size k in reduced G;
5. **for** each C **do**
6. for each vertex v in C, produce a copy vertex v';
7. Construct a bipartite graph $G' = (V_1 \cup V_2, E)$ in the following way: the edges connecting to v are also connected to v'. The k vertices and its copy vertices are put into V_1, and the neighbors of the k vertices in C are put into V_2;
8. use Maximum bipartite matching algorithm to find the k center vertices;
9. **if** all the vertices on V_1 are matched **then**
   report "there exists a k-P_2-packing in G" and stop;
10. report "there is no a k-P_2-packing in G" and stop;

Fig. 4. KPPW agorithm

Theorem 4. *If there exists a k-P_2-packing, the KPPW algorithm will find the k-P_2-packing in time $O^*(2^{4.142k})$.*

Proof. It can be seen from the algorithm, the step2 is the whole kernelization process applying the RPLW algorithm repeatedly. As a result, we can obtain a kernel of size at most $7k$. We check the number of vertices in reduced G in Step3. If the number of vertices is more than $7k$, there must be a k-P_2-packing in G, and the KPPW algorithm does not need to run. We apply a straightforward brute-force method on the kernel to find the optimal solution from Step 4 to Step 8. The general idea is as follows:

We find all possible subsets C of size k in reduced graph G. For each C, we will construct a bipartite graph $G' = (V_1 \cup V_2, E)$ in the following way: first, for each vertex v in C, we produce a copy vertex v' with the property that the edges connecting to v are also connected to v'. The k vertices and its copy vertices are put into V_1, and the neighbors of the k vertices in C are put into V_2. If all the vertices on the V_1 are matched by a maximum bipartite matching, the k vertices in C must be the center vertices of a k-P_2-packing, therefore, report "there exists a k-P_2-packing in G" , and the algorithm stops. If for all C, we cannot find k center vertices, report "there does not exist a k-P_2-packing in G", and the algorithm stops.

In the following, we analyze the time complexity of algorithm KPPW.

Step1: Using greedy algorithm to find a maximal packing can be done in time $O(|E|)$.

Step2: The kernelization process given in [7] can be done in $O(n^3)$ time, and the whole process that call the algorithm RPLW repeatedly until Rule1 and Rule2 are not applicable runs in $O(k^3)$ time, therefore, the time complexity of Step2 is $O(n^3 + k^3)$.

Step3: Obviously, the running time of Step 3 is linear in the size of V.

Step4-Step8: We find the center vertices of the P_2-packing in a brute force manner, which has $\binom{7k}{k}$ enumerations. By Stirling's formula, this is bounded by $2^{4.142k}$. We construct a bipartite graph $G' = (V_1 \cup V_2, E)$ with k vertices in C and its copy vertices in V_1, and the neighbors of k vertices in V_2. Thus, the original question is transformed to find the maximum matching problem in bipartite G' which can be solved in time $O(\sqrt{|V_1 + V_2|}|E|) = O(k^{2.5})$. Therefore, the total running time of Step4-Step8 is $O(2^{4.142k}k^{2.5})$.

As a result, the total running time of algorithm KPPW is bounded by $O(|E| + |n^3 + k^3 + |V| + k + 2^{4.142k}k^{2.5}) = O^*(2^{4.142k})$. □

5 Conclusions

In this paper, we mainly focus on the kernelization for the k-P_2-packing problem. We give further structure analysis of the problem, and propose a kernelization algorithm obtaining a kernel of size at most $7k$. Comparing with the kerneliztion given in [7], our algorithm makes further optimization on the vertices of any P_2 in W and their Q_0-vertex neighbors and Q_1-edge neighbors, which reduces the number of Q_0-vertices and Q_1-edges in Q. Based on the $7k$ kernel, we also present an improved parameterized algorithm with time complexity $O^*(2^{4.142k})$, which greatly improves the current best result $O^*(2^{5.301k})$.

References

1. Hell, P.: Graph Packings. Electronic Notes in Discrete Mathematics 5 (2000)
2. Kirkpatrick, D.G., Hell, P.: On the complexity of general graph factor problems. SIAM J. Comput. 12, 601–609 (1983)
3. Kann, V.: Maximum bounded H-matching is MAX-SNP-complete. J. nform. Process. Lett. 49, 309–318 (1994)
4. Hurkens, C., Schrijver, A.: On the size of systems of sets every t of which have an SDR, with application to worst case ratio of Heuristics for packing problems. ISIAM J. Discrete Math. 2, 68–72 (1989)
5. Fellows, M., Heggernes, P., Rosamond, F., Sloper, C., Telle, J.A.: Exact algorithms for finding k disjoint triangles in an arbitrary graph. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 235–244. Springer, Heidelberg (2004)
6. Mathieson, L., Prieto, E., Shaw, P.: Packing edge disjoint triangles: a parameterized view. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 127–137. Springer, Heidelberg (2004)
7. Prieto, E., Sloper, C.: Look at the stars. Theoretical Computer Science 351, 437–445 (2006)
8. Chen, J.: Parameterized computation and complexity: a new approach dealing with NP-hardness. Journal of Computer Science and Technology 20, 18–37 (2005)