The problem of packing modification-disjoint P_3 – an overview and an improved heuristic approach

Jona Dirks[1], Enna Gerhard[2]

Abstract: A P_3 is a path on three vertices. Two P_3 are *modification-disjoint* if they share at most one vertex. In this work we consider the problem of packing modification-disjoint induced P_3.

To the best of our knowledge it is not known whether the problem of finding a maximum packing of modification-disjoint P_3 can be solved in polynomial time. We provide new insights towards this question.

We provide an overview and new insights for locating modification-disjoint P_3 packing within the complexity hierarchy. Accordingly, we further look into conflict graphs.

Complementing our theoretical results, we present a significantly improved heuristic based on the approach of Spinner [Sp19]. We analyze its efficiency empirically on a selection of generated and public datasets. Our results improve on existing heuristics when comparing solution size and running time.

Keywords: P3 Packing; Cluster Editing; Graph Theory; Heuristic; Complexity

1 Introduction

A P_3 is an induced path on three vertices. Two P_3 are called modification-disjoint if they share at most one vertex. The modification-disjoint P_3 packing problem (PPP) is to find a largest set of pairwise modification-disjoint P_3 within a given graph.

To the best of our knowledge, PPP has not been studied in the literature. It is not even known whether PPP is solvable in polynomial time.

PPP is very relevant for the cluster editing problem (CEP). The CEP is the problem of finding a $M \subseteq E$ such that $(V, E \Delta M)$ is a cluster graph with all components being cliques such that $|M| \leq k$. PPP can be used to find a lower bound for CEP, which can be applied in branching algorithms. CEP has many applications, for example in bioinformatics, image processing and circuit design, see e.g. [SST04]. Solving CEP efficiently was the goal of the PACE 2021 Challenge in which we submitted the best student solver [Di21b].

[1] University of Bremen, Faculty 3, Bibliothekstraße 1, 28334 Bremen, Germany
dirks2@uni-bremen.de
[2] University of Bremen, Faculty 3, Bibliothekstraße 1, 28334 Bremen, Germany
gerhard@uni-bremen.de

After introducing some notation in Sect. 2, we discuss results regarding the complexity of PPP in Sect. 3. One possible way to show complexity bounds could be to understand the graph class of conflict graphs. We will study these in Sect. 3.2. Afterwards, we use proven characteristics of these conflict graphs to improve upon known heuristics in Sect. 4.

2 Definitions and notation

All graphs in this paper are undirected and do not contain double edges or loops. We use standard graph notation. We write ab for an edge between vertices a and b. A $K_{a,b}$ is the complete bipartite graph partitioned into a and b vertices. W_6 is the six wheel. See Fig. 1.

We represent a P_3 via the set of vertices that it contains. For simplicity such a set over the vertices a, b and c is denoted as abc. We require that P_3 are induced. In most cases we also assume that the non-edge is ac. The set of all induced P_3 of a graph G is denoted with $P_3(G)$, see Fig. 2 for P_3 (W_6).

Two P_3, abc and def are modification-disjoint if they share one or less vertices, that is, if $|abc \cap efd| \leq 1$. In this case, they share neither an edge ab or bc nor the non-edge ac. We say that two P_3 are in conflict, if they are not modification-disjoint. The conflict graph for a graph G is induced by the edges $\{pq \mid p, q \in P_3(G), p \neq q, p \text{ in conflict with } q\}$. In case of the six wheel W_6, this becomes a petersen graph, see Fig. 3.

The class of *F-free graphs* for a set of graphs F is the set of graphs that do not contain any graph from F as induced subgraphs. A *cluster graph* is a graph of fully connected components. The class of cluster graphs is exactly the class of P_3-free graphs.

3 Observations regarding the complexity of PPP

This section discusses the decision problem variant DPPP where (G, k) is a yes-instance, if there exists a modification-disjoint P_3 packing in G of size at least k. To the best of our knowledge, it is yet unknown whether DPPP is NP-complete or solvable in polynomial time. There are similar problems that are within either of these classes. The most promising approach for solving PPP is a reduction to independent set on the restricted graph class of conflict graphs which we will explore in Sect. 3.2.

3.1 Related problems

There are several related problems for which the complexity is known. For example if we ignore the non edge and only search for edge disjoint P_3 packings, Algorithm 1 runs in polynomial time [DS21]. A maximum matching in G can be computed using the Blossom-Algorithm by Edmonds [Ed65].

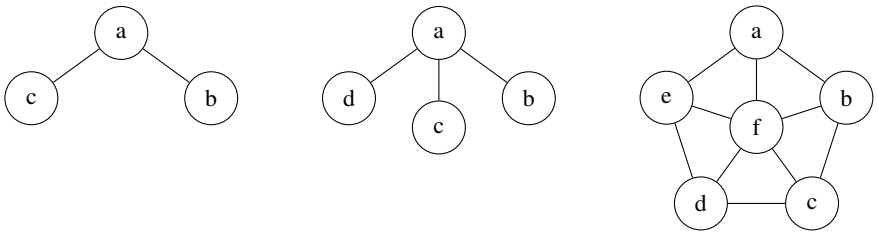

Fig. 1: $K_{1,2}$ containing one induced P_3, the claw $K_{1,3}$ and the six wheel W_6

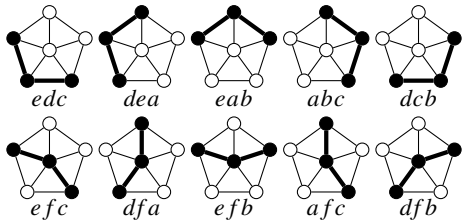

Fig. 2: P_3 (W_6), all induced P_3 on the six wheel W_6

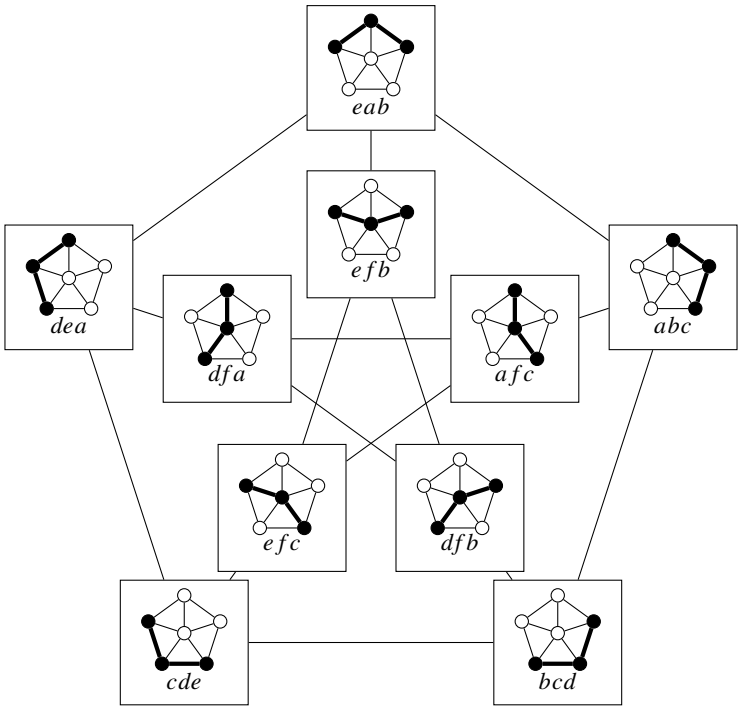

Fig. 3: Conflict graph of W_6

Algorithm 1: Polynomial ($O(|E|^4)$) time edge disjoint P_3 packing algorithm

Input: Graph G, size k
Output: A packing of size k exists
$EG := (E(G), \{e_1e_2 \mid e_1, e_2 \in E(G) \text{ and } e_1 \text{ adjacent to } e_2\})$ ▷ The edge graph
for *each* $e_1e_2 \in E(EG)$ **do**
  if $e_1 \cup e_2 \notin P_3(G)$ **then**
    Remove edge e_1e_2 from EG
$S := \emptyset$
for *each matching* $\{e_1, e_2\} \in$ *maximumMatching(EG)* **do**
  add $e_1 \cup e_2$ to S ▷ The set $e_1 \cup e_2$ induces a P_3
return $|S| \geq k$

On the other hand, if we require that all vertices have to be covered by vertex-disjoint P_3, then the problem becomes NP-complete [Pa94].

3.2 Characteristics of conflict graphs

Observation 3.1. *A maximum modification-disjoint P_3 packing corresponds directly to an independent set in the conflict graph.*

While the independent set problem is NP-complete in general, it can be solved efficiently in many restricted graph classes, e.g. in $K_{1,3}$-free (claw-free) graphs. This motivates our study of the class of conflict graphs. A very interesting property of conflicts graphs is the neighbour covering of cliques which can be formulated as:

Theorem 3.2. *For each vertex v of a conflict graph C of G there exist $Q, R, S \in clique(v)$ such that each $u \in N(v)$ lies in $Q \cup R \cup S$. Here, $clique(v)$ denotes the set of all cliques which contain v.*

Proof. For each vertex in C there exists a P_3 ijk in G. This P_3 has three possible conflicts. If we collect all P_3 that are in conflict via a (non-)edge ab into one set $[ab]$, then this P_3 has three such sets, namely $[ij]$, $[jk]$ and $[ki]$. The P_3 of each set form a clique in the conflict graph C, because they are all in conflict via the name giving edge. Therefore, we can choose these sets as the cliques Q, R and S [Di21a]. □

We next prove that conflict graphs cannot be described as F-free for any set F. This is unfortunate, as F-free graphs are well studied. For example if conflict graphs would be $K_{1,3}$-free, which they are not, then DPPP would be solvable in polynomial time due to the mentioned connection with independent set [Sb80]. On the other hand, if they were characterized as the class of $K_{1,4}$-free graphs, then independent set would be NP-complete on them [Mi80]. To prove this, we first show inclusion and exclusion of certain subgraphs as conflict graphs:

Lemma 3.3. *If a graph G has two not necessarily induced P_3 abc and adc then $abc \in P_3(G) \Leftrightarrow adc \in P_3(G)$.*

Proof. It is obvious that we only have to show one implication, because the other one is analog. Let $abc \in P_3(G)$. If adc was not induced, then there would have to be an edge ac. The P_3 abc would not be induced either. □

Lemma 3.4. *$K_{1,3}$ is not a conflict graph.*

Proof. We can show that there is just a single possibility for a $K_{1,2}$ to be present without structures that are in conflict with a $K_{1,3}$. We do so by looking at three cases with subcases that are visualized in Fig. 4. We then show that there is no structure that leads to a $K_{1,3}$.

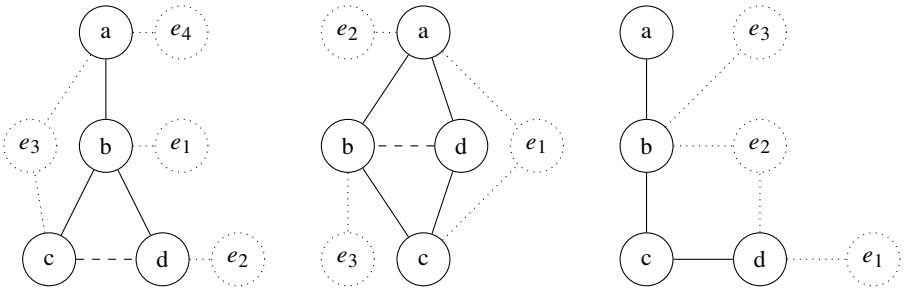

Fig. 4: The possible ways of extending conflict graphs with a $K_{1,1}$

Case 1. There are two P_3 in conflict via one edge. So without loss of generality, $abc, abd \in P_3(G)$. Then there also has to be an edge dc, otherwise the conflict graph would contain a triangle. Now there are four different possibilities that potentially lead to a $K_{1,2}$.

Case 1.1. There exists a P_3 abe_1. This P_3 is in conflict with both abc and abd. Therefore, the resulting graph can never be a $K_{1,3}$, even if further edges or vertices are present.

Case 1.2. There exists a P_3 e_2db (the case e_2cb is analog). To avoid the P_3 e_2dc, there has to be an edge e_2c. However, the P_3 e_2cd must exist – because of Lemma 3.3 the P_3 e_2db would not be possible otherwise.

Case 1.3. There exists a P_3 ae_3c (the case de_3a is analog). To avoid the P_3 bae_3, the edge be_3 has to exist. Now the edge de_3 has to exist to avoid the P_3 dbe_3. Because of Lemma 3.3, the P_3 de_3a must be present, otherwise abd could not exist.

Case 1.4. There exists a P_3 e_4ab. In this case, it is possible to construct a $K_{1,2}$. We will discuss this case at the end of the proof.

Case 2. There are two P_3 in conflict via the non-edge. Without loss of generality, $abc, adc \in P_3(G)$. Then there also has to be an edge bd. Now there are three different possibilities that potentially lead to a $K_{1,2}$.

Case 2.1. There exists a P_3 ae_1c. This P_3 is in conflict with both abc and adb. Therefore, the resulting graph can never be a $K_{1,3}$, even if further edges or vertices are present.

Case 2.2. There exists a P_3 e_2ab (the case e_2cd is analog). To avoid the P_3 e_2ad the edge de_2 has to exist. Because of Lemma 3.3, the P_3 bde_2 must be present, otherwise the P_3 e_2ab could not exist.

Case 2.3. There exists a P_3 e_3ba (the case e_3da is analog). To avoid the P_3 dbe_3 the edge de_3 has to be present. Because of Lemma 3.3, the P_3 e_3da must be present, otherwise the P_3 e_3ba could not exist.

Case 3. $abc, bcd \in P_3(G)$

Case 3.1. There exists a P_3 cde_1 (the case e_1ab is analog). This would produce a P_3 as a conflict graph, which cannot be part of a $K_{1,3}$.

Case 3.2. The P_3 be_2d or ae_2c exist. This case is analog to Case 2.2.

Case 3.3. The P_3 abe_3 or dce_3 exist. This case is analog to Case 1.4.

We return to Case 1.4. To extend the conflict graph of it to a $K_{1,3}$ the new P_3 has to contain e. Otherwise, a $K_{1,2}$ would exist, which has been covered above. We can now look at three cases where another P_3 could appear, which can be seen in Fig. 5.

Case 1. There exists a P_3 aef_1. This would produce a path of length 4 (P_4) as a subgraph in the conflict graph.

Case 2. There exists a P_3 f_2ae. To avoid a P_3 baf_2, there have to be edges bf_2, cf_2 and df_2. Now because of Lemma 3.3, the P_3 df_2a cannot be avoided with abd present.

Case 3. There exists a P_3 bf_3e. We then need edges af_3, cf_3, df_3. Now we cannot avoid the P_3 df_2a with abd existing.

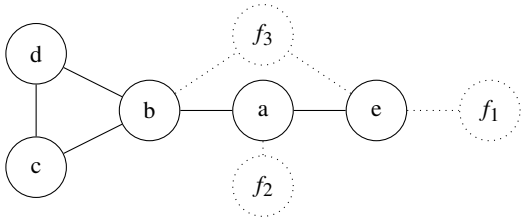

Fig. 5: The possible ways of extending a $K_{1,2}$ □

However this does not rule out a $K_{1,3}$ as an induced subgraph:

Lemma 3.5. *There are non-$K_{1,3}$-free conflict graphs.*

Proof. The conflict graph of W_6 (Fig. 3) has the induced $K_{1,3}$ over the vertices efb, dfb, eab and efc. □

With this we can prove the following theorem:

Theorem 3.6. *Conflict graphs cannot be described as F-free for any graph set F.*

Proof. Assume towards a contradiction that there is an F such that conflict graphs are exactly the class of F-free graphs. By definition of the induced subgraph relation any induced subgraph of a conflict graph has to be also F-free. Because of Lemma 3.5 the graph $K_{1,3}$ is then also F-free. This would mean that $K_{1,3}$ is a conflict graph, which was disproved in Lemma 3.4. □

3.3 Reduction to independent set on conflict graphs

We know that if we can solve independent set on the restricted graph class of conflict graphs in polynomial time, we can also solve DPPP. Unfortunately, as we have seen in the previous section, this graph class is hard to describe and does not fit in the well studied concept of F-free graphs. Even intuitively, this problem is hard to narrow down, as it is similar both to problems that are solvable in polynomial time and problems that are NP-complete.

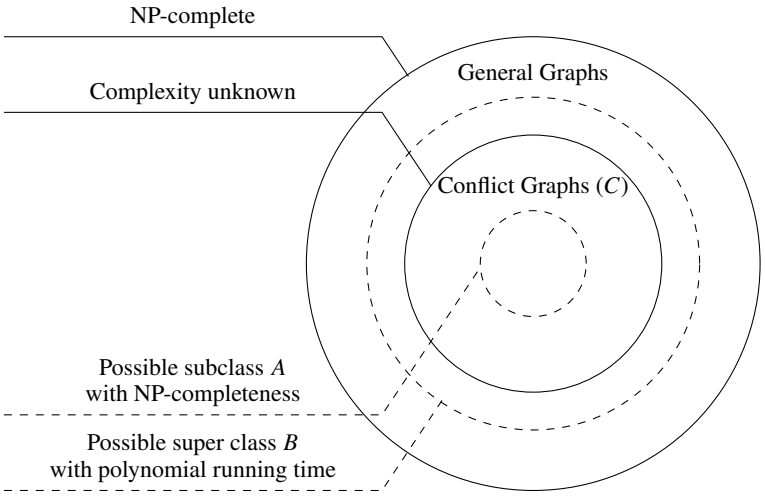

Fig. 6: Complexity of the independent set problem on different graph classes. By Theorem 3.6 we get $A \neq C \neq B$, if A and B are described as F-free

There might be a subclass of conflict graphs where solving independent set is NP-complete or a super class for which a polynomial time algorithm exists. This relationship can be seen in Fig. 6. Independent set remaining NP-hard on conflict graphs would not imply the same

for DPPP as we haven't been able to reconstruct the original graph for a conflict graph in the general case. As a result, it is still unknown to which complexity class this problem belongs.

4 Heuristic approaches to PPP

As the complexity of PPP remains unknown, our best possible approach is to use heuristics. In this section, we will introduce known heuristics and show that with our results about conflict graphs we are able to improve upon these.

4.1 Known heuristics

A possible way to solve this problem is to reduce it to an independent set instance and then solve this one. One such local search heuristic was developed by Nogueira et al. [NPS18]. The heuristic is based on swaps: An (a,b)-$swap$ removes a vertices form the solution and adds b to the solution. The heuristic searches for $(w,1)$-$swap$s and $(1,2)$-$swap$s. The first case can only happen in weighted instances and thus is left out in our adaptation. The starting solution is found greedily. Nogueira et al.'s heuristic also makes a perturbation step that modifies the solution. These steps are repeated until some termination criteria are met.

Spinner later adopted this heuristic to find packings for various graphs [Sp19]. However, for some unknown reason he does not make use of the perturbation step and also simplified the termination criteria. In our implementation the $(w,1)$-$swap$ was also left out.

4.2 Novel heuristic

Especially the results from Theorem 3.2 allow us to improve on Spinners heuristic. The algorithm calculates these cliques while searching for the set of P_3. It can then compute possible $(1,2)$-$swap$s faster, because any two vertices of a swap have to come from two different cliques in the conflict graph. While in theory such a swap can be found in any combination of cliques, we decided to ignore the one that is produced by the non-edge, as this did not improve results.

The resulting practical time improvement allows us to include other attempts to increase solution size while still being faster than the heuristic developed by Spinner:

- If we found a $(1,2)$-$swap$, we can try to improve it to a $(1,3)$-$swap$.
- We attempt applying non-improving $(1,1)$-$swap$s, at least once, or as long as further improvements are found. This is a minimalistic version of the perturbation step done by [NPS18].

- We can search for simplicial vertices and force add them into the solution. A vertex is called simplicial, if its neighbors form a clique. This is based on a reduction rule developed by [St16]. Theorem 3.2 allows us to check whether all but one of the cliques are empty.

The full heuristic in pseudocode is shown in Algorithm 2. A $(1,3)$- or $(1,2)$-$swap$ can be calculated with Algorithm 3. A $(1,1)$-$swap$ is trivial, it replaces a P_3 with a different one, thus retaining solution size. The greedy packing can be calculated by iterating through all triples of vertices and adding them to P if they are a P_3 and not in conflict with any previously added P_3. The cliques from Theorem 3.2 ($[ab]$, $[bc]$, $[ca]$ for P_3 abc) can also be calculated in this step. A further improvement to running time in practise is to keep track of the number of P_3 in P that any P_3 is in conflict with.

Algorithm 2: Improved heuristic

```
Input: A Graph G
Output: Packing P
Function isSimplicial(p) is
    return |[ab]| > 0 and |[bc]| = 0 and |[ca]| = 0 for pairwise unequal a, b, c ∈ p
P := greedyPacking(G)
fixed(p) := isSimplicial(p), ∀p ∈ P
for each p ∈ (P_3(G) \ P) with isSimplicial(p) do
    fixed(p) := true
    remove all [ab] from P
attempt := 0
while attempt ≤ 1 do
    attempt += 1, P_last := ∅
    while |P_last| < |P| do
        P_last := P
        S := swapSearch(P, fixed)                    ▷ Algorithm 3
        if S exists then
            apply S to P
            attempt := 0
    S := find (1, 1)-swap
    if S exists then
        apply S to P
    else
        return P
return P
```

We obtain a worst case running time of $O(|P_3(G)|^4) \cdot O(|P_3(G)|) = O(|P_3(G)|^5)$, where the first factor stems from Algorithm 2 and the second from Algorithm 3. The amount of $P_3(G)$ can be at most $|V|^3$. This is a worst case estimate and there are hidden factors that drastically decrease running time in practice. If we have large cliques in the conflict graph and thus a high running time of Algorithm 3, we typically obtain a small maximum packing and therefore a smaller amount of iterations in Algorithm 2 will be sufficient.

Algorithm 3: Swap search for $(1,2)$-$swaps$ and $(1,3)$-$swaps$

Input: A packing P, status $fixed$
Output: A suggested swap, if one exists
Function *conflict(p, P)* **is**
  return $|\{p' \mid p' \in P : p' \text{ is in conflict with } p\}|$
for *each* $ijk \in P$ *and* $\neg fixed(ijk)$ **do**
  for *each* $a \in [ij]$, a, $conflict(p, P) \leq 1$ *and* $\neg fixed(a)$ **do**
    for *each* $b \in [jk]$, $conflict(a, P) \leq 1$ *and* $conflict(b, \{a\}) = 0$ *and* $\neg fixed(b)$ **do**
      for *each* $c \in [ik]$, $conflict(c, P) \leq 1$ *and* $conflict(c, \{a, b\}) = 0$ *and* $\neg fixed(c)$ **do**
        return swap ijk for a, b, c
      return swap ijk for a, b
return *no swap exists*

4.3 Empiric results

To show that our heuristic performs better than previously known heuristics, we evaluated it on two types of graphs. First we used randomly generated relatively small graphs in order to show the relation to the slow heuristics. Secondly we used the smallest 50 public instances of the PACE Challenge [Ke21]. We generated 50 graphs of the sizes $n = 10, m = 20$ (small), $n = 20, m = 80$ (medium), $n = 20, m = 100$ (large) and $n = 40, m = 300$ (huge).

The results are shown in Tab. 1. For evaluating the running time we used an AMD Ryzen 5 3600 6-Core processor with 16 gigabyte of RAM. For the ILP formulation we used a translation of the folklore ILP for independent set on the conflict graph: $\max \sum_{p \in P_3(G)} x_p$ such that $(x_p + x_{p'} \leq 1, \forall p, p' \in P_3(G) : p \text{ in conflict with } p')$ and $(x_p \in \{0, 1\}, \forall p \in P_3(G))$ (see [BB13]).

The greedy heuristic is fastest, but its solution sizes get outperformed on all but the smallest generated graphs. While Nogueira et al. produces almost perfect solutions, which compare to the exact ILP, it is very slow. On most graph types it is even slower than the ILP. We were not able to compute Nogueira nor the ILP on the huge graphs of the PACE Challenge within reasonable time. Thus, both algorithms seem too slow to solve PPP on real world instances.

With respect to the running time, the novel heuristic is faster on all graphs compared to Spinner. On the large graphs the novel heuristic has a speedup of a magnitude of at least ten. On the pace graphs this time advantage is only around 10%, although it is still clearly visible.

Regarding the packing size, the results are not so clear but still indicate a lead for the novel heuristic. In two categories the novel heuristic produces larger packings, otherwise it performed equally well.

However, the PACE dataset is highly irregular, and a simple average hides this. There is more than a 100-fold increase in edge count between the smallest and the largest graphs. A

Tab. 1: Empirical results, average over tested instances of the respective sizes

Graph Type	Solution size				
	Greedy	Spinner	Novel	Nogueira	ILP
small	8	8	9	9	9
medium	37	38	39	39	40
large	45	47	48	49	50
huge	146	149	149	150	150
pace	677	712	712	timeout	timeout

Graph Type	Running time in milliseconds				
	Greedy	Spinner	Novel	Nogueira	ILP
small	0	2	0	54	11
medium	0	86	6	2074	271
large	0	142	10	3395	528
huge	0	2449	444	63354	63978
pace	19	400903	363999	timeout	timeout

small disadvantage on one of the larger graphs could cover advantages on many smaller graphs. The novel heuristic has better results on 23 graphs while Spinners heuristic performs better on 21 graphs, giving the novel heuristic a small lead. The novel heuristic is faster on all but three graphs regarding the running time.

5 Conclusion and further research

We have explored PPP and explained several of its properties.

There are many similar problems that are NP-complete or in P. The exact classification of PPP remains interesting for further research.

We have found quite a few characteristics about conflict graphs, such as the partition of neighbours of each vertex into three cliques. Although we were only able to show that conflict graphs can not be described with commonly used techniques, we have added insights for further developments. We were able to use our results to improve upon know heuristics both in terms of speed and of solution size, and we were able to demonstrate these gains.

References

[BB13] Böcker, S.; Baumbach, J.: Cluster Editing. In (Bonizzoni, P.; Brattka, V.; Löwe, B., eds.): The Nature of Computation. Logic, Algorithms, Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 33–44, 2013, ISBN: 978-3-642-39053-1, URL: https://doi.org/10.1007/978-3-642-39053-1.

[Di21a] Dietrich, K.; Dirks, J.; Gahde, J.; Heydt, O.; Schnaubelt, Y.; Sczuka, F.; Siering, N.; Sonneborn, M.; Stichternath, L.; Tat, J.; Freese, T.: Parametrisierte Algorithmen auf Graphen, Project Report, Bremen, Germany, July 2021.

[Di21b] Dirks, J.; Grobler, M.; Rabinovich, R.; Schnaubelt, Y.; Siebertz, S.; Sonneborn, M.: PACE Solver Description: PACA-JAVA. In (Golovach, P. A.; Zehavi, M., eds.): 16th International Symposium on Parameterized and Exact Computation (IPEC 2021). Vol. 214, Dagstuhl, Germany, 30:1–30:4, 2021.

[DS21] Dirks, J.; Schnaubelt, Y.: Modellbasierte Parameterkonfiguration für parametrisierte Graphprobleme, (Sieberts, S.; Wright, M., supv.), Bachelor Thesis, Bremen, Germany: University of Bremen, Sept. 2021.

[Ed65] Edmonds, J.: Paths, Trees, and Flowers. Canadian Journal of Mathematics 17/1, pp. 449–467, 1965, URL: https://doi.org/10.4153/CJM-1965-045-4.

[Ke21] Kellerhals, L.; Koana, T.; Nichterlein, A.; Zschoche, P.: The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In. Vol. 214, 2021, URL: https://doi.org/10.4230/LIPIcs.IPEC.2021.26.

[Mi80] Minty, G. J.: On maximal independent sets of vertices in claw-free graphs. Journal of Combinatorial Theory, Series B 28/3, pp. 284–304, 1980, ISSN: 0095-8956, URL: https://doi.org/10.1016/0095-8956(80)90074-X.

[NPS18] Nogueira, B.; Pinheiro, R. G. S.; Subramanian, A.: A hybrid iterated local search heuristic for the maximum weight independent set problem. Optimization Letters 12/3, pp. 567–583, 2018, URL: https://doi.org/10.1007/s10878-014-9756-7.

[Pa94] Pantel, S.: Graph Packing Problems, (Hell, P.; Alspach, B.; Brewster, R., supv.), Master Thesis, Manitoba, Canada: University of Manitoba, Oct. 1994.

[Sb80] Sbihi, N.: Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoilé. French, Discrete Math. 29/, pp. 53–76, 1980, ISSN: 0012-365X, URL: https://doi.org/0012-365X(90)90287-R.

[Sp19] Spinner, J.: Weighted F-free Edge Editing, (Hamann, M.; Gottesbüren, L., supv.), Bachelor Thesis, Karlsruhe, Germany: Karlsruhe Institute of Technology, 2019, URL: https://doi.org/10.5445/IR/1000135117.

[SST04] Shamir, R.; Sharan, R.; Tsur, D.: Cluster graph modification problems. Discrete Applied Mathematics 144/1, Discrete Mathematics and Data Mining, pp. 173–182, 2004, ISSN: 0166-218X, URL: https://doi.org/10.1016/j.dam.2004.01.007.

[St16] Strash, D.: On the Power of Simple Reductions for the Maximum Independent Set Problem. In (Dinh, T. N.; Thai, M. T., eds.): Computing and Combinatorics. Springer International Publishing, Cham, pp. 345–356, 2016, ISBN: 978-3-319-42634-1, URL: https://doi.org/10.1007/978-3-319-42634-1_28.