Finding k disjoint triangles in a graph*

Mike Fellows† Pinar Heggernes‡ Frances Rosamond†
Christian Sloper‡ Jan Arne Telle‡

Abstract

We consider exact algorithms for the *NP*-complete problem of deciding whether an input graph on n vertices has k vertex-disjoint copies of a fixed graph H. For $H = K_3$ (the triangle) we give an $O(2^{2k \log k + 1.869k} n^2)$ algorithm, and for general H an $O(2^{k|H| \log k + 2k|H| \log |H|} n^{|H|})$ algorithm. We introduce a preprocessing (kernelization) technique based on crown decompositions of an auxiliary graph. For $H = K_3$ this leads to a preprocessing algorithm that reduces an arbitrary input graph of the problem to a graph on $O(k^3)$ vertices in polynomial time.

1 Introduction

For a fixed graph H and an input graph G, the H-packing problem asks for the maximum number of vertex-disjoint copies of H in G. The K_2-packing problem, which is equivalent to maximum matching, played a central role in the history of classical computational complexity. The first step towards the dichotomy of "good" (polynomial-time) versus "presumably-not-good" (*NP*-hard) was made in a paper on maximum matching from 1965 [E65], which gave a polynomial time algorithm for that problem. On the other hand, the K_3-packing problem, which is our main concern in this paper, is *NP*-hard [HK78].

Recently, there has been a growing interest in the area of exact exponential-time algorithms for *NP*-hard problems. When measuring time in the classical way, simply by the size of the input instance, the area of exact algorithms for *NP*-hard problems lacks the classical dichotomy of good (P) versus presumably-not-good (*NP*-hard) [W03]. However, if in the area of exact algorithms for *NP*-hard problems we instead measure time in the parameterized way, then we retain the classical dichotomy of good (FPT - Fixed Parameter Tractable) versus presumably-not-good ($W[1]$-hard) [DF99]. It therefore seems that the parameterized viewpoint gives a richer complexity framework. In fact, a formal argument for this follows from the realization that the non-parameterized viewpoint, measuring time by input size, is simply a special case of the parameterized viewpoint with the parameter chosen to be the input size. Parameterized thusly, any problem is trivially FPT and the race for the best FPT algorithm is precisely the same as the race for the best non-parameterized exact algorithm. Note that for any optimization or decision problem, there are many interesting possibilities for choice of parameter, that can be guided by both practical and theoretical considerations, see for example [F03] for a discussion of five different parameterizations of a single problem. In our opinion, the relevant discussion for the field of exact algorithms

*This work was initiated while the first and third authors were visiting the University of Bergen
†School of Electrical Engineering and Computer Science, University of Newcastle, Australia. Email: {mfellows, fran}@cs.newcastle.edu.au
‡Department of Informatics, University of Bergen, Norway. Email: {pinar, sloper, telle}@ii.uib.no

for *NP*-hard problems is therefore not "parameterized or non-parameterized?" but rather "which parameter?"

In this paper our focus is on parameterized exact algorithms for deciding whether a graph G has k disjoint copies of K_3, with the integer k being our parameter. On input (G, k), where G is a graph on n vertices, an FPT algorithm for this problem is one with runtime $O(n^{\alpha} f(k))$, for a constant α and an unrestricted function $f(k)$. We want, of course, both α and the growth rate of $f(k)$ to be as small as possible.

A practical spinoff from the field of parameterized exact algorithms for *NP*-hard problems has been a theoretical focus on the algorithmic technique of preprocessing, well-known from the heuristic algorithms community. In fact, the parameterized problems having FPT algorithms are *precisely* the parameterized problems where preprocessing can in polynomial time reduce a problem instance (G, k) to a kernel, *i.e.*, a decision-equivalent problem instance (G', k') where the size of G' is bounded by a function of k (only), and where also $k' \leq k$ [DFS97]. One direction of this fact is trivial, since any subsequent brute-force algorithm on (G', k') would give an overall FPT algorithm. In the other direction, assume we have an FPT algorithm with runtime $O(n^{\alpha} f(k))$ and consider an input (G, k) on n vertices. If $n \geq f(k)$ then the runtime of the FPT algorithm on this instance is in fact polynomial and can be seen as a reduction to the trivial case. On the other hand, if $n \leq f(k)$ then the instance (G, k) already satisfies the kernel requirements. Note that in this case the kernel size $f(k)$ is exponential in k, and a smaller kernel is usually achievable. For this reason, in the field of parameterized algorithms for *NP*-hard problems, it can be argued that there are two distinct races [F03]:

- Find the fastest FPT algorithm for the problem.
- Find the smallest polynomial-time computable kernelization for the problem.

In this paper, we enter the parameterized K_3-packing problem into both these races, giving on the one hand an $O(2^{2k \log k + 1.869k} n^2)$ FPT algorithm, and on the other hand an $O(k^3)$ kernelization. Our FPT algorithm is derived by an application of a fairly new technique known as greedy localization [JZC04], and our kernelization algorithm by a non-standard application of the very recently introduced notion of Crown Reduction Rules [CFJ03, F03]. We end the paper by asking how well these two results on K_3-packing generalize to H-packing. It turns out that the FPT algorithm generalizes quite easily, giving FPT algorithms for deciding whether an input graph G has k disjoint copies of an arbitrary connected H. However, we presently do not see how to generalize the kernelization algorithm.

The next section gives some basic graph terminology. We then proceed in Sections 3, 4 and 5 with the kernelization results, before continuing with the FPT algorithm in Section 6 for K_3 and in Section 7 for general H.

2 Preliminaries

We assume simple, undirected, connected graphs $G = (V, E)$, where $|V| = n$. The neighbors of a vertex v are denoted by $N(v)$. For a set of vertices $A \subseteq V$, $N(A) = \{v \notin A \mid uv \in E$ and $u \in A\}$, and the subgraph of G induced by A is denoted by $G(A)$. For ease of notation, we will use informal expressions like $G \setminus u$ to denote $G(V \setminus \{u\})$, and $G \setminus e$ to denote $(V, E \setminus \{e\})$, where u is a vertex and e is an edge in G. A subset S of V is a *separator* if $G \setminus S$ is disconnected.

An *H-packing* W of G is a collection of disjoint copies of graph H in G. We will use $V(W)$ to denote the vertices of G that appear in W, and $E(W)$ to denote the edges. A *matching* is a K_2-packing.

We will in the following two sections describe a set of reduction rules. If any of these rules can be applied to G, we say that G is *reducible*, otherwise *irreducible*.

3 Reduction rules for K_3-packing

Let us start with a formal definition of the problem that we are solving:

> k-K_3-PACKING (TRIANGLE PACKING)
> INSTANCE: Graph $G = (V, E)$
> PARAMETER: k
> QUESTION: Does G have k disjoint copies of K_3?

We say that a graph G has a k-K_3-packing if the answer to the above question is, "yes." In this section, we identify vertices and edges of the input graph that can be removed without affecting the solution of the k-K_3-PACKING problem.

Definition 1 *If vertices a, b, and c induce a K_3, we say that vertex a* sponsors *edge bc. Likewise, edge bc* sponsors *vertex a.*

We start with two simple observations that also give preprocessing rules useful to delete vertices and edges that cannot participate in any triangle.

Reduction Rule 1 *If $e \in E$ has no sponsor then G has a k-K_3-packing $\Leftrightarrow$ $G \setminus e$ has a k-K_3-packing.*

Reduction Rule 2 *If $u \in V$ has no sponsor then G has a k-K_3-packing $\Leftrightarrow$ $G \setminus u$ has a k-K_3-packing.*

Both observations are trivially true, and let us remove vertices and edges from the graph so that we are left with a graph containing only vertices and edges that could potentially form a K_3.

Reduction Rule 3 *If $u \in V$ sponsors at least $3k - 2$ disjoint edges then G has a k-K_3-packing $\Leftrightarrow$ $G \setminus u$ has a $(k-1)$-K_3-packing.*

Proof. ($\Rightarrow$:) This direction is clear as removing one vertex can decrease the number of K_3s by at most one.
($\Leftarrow$:) If $G \setminus u$ has a $(k-1)$-K_3-packing S, then S can use vertices from at most $3(k-1) = 3k-3$ of the disjoint edges sponsored by u. This leaves at least one edge that can form a K_3 with u, thus raising the number of K_3s to k. $\square$

4 Reducing independent sets - crown reduction

Motivated by the following simple reduction rule, we will in this section give a generalization of it.

Reduction Rule 4 *If $\exists u, v \in V$ such that $N(u) = N(v) = \{a, b\}$ and $ab \in E$ then G has a k-K_3-packing $\Leftrightarrow$ $G \setminus u$ has a k-K_3-packing.*

Proof. This is trivial as it is impossible to use both u and v in any K_3-packing. $\square$

This reduction rule identifies a redundant vertex and removes it. The vertex is redundant, because it has a stand-in that can form a K_3 in its place and there is no use for both vertices. Generalizing, we try to find a set of vertices such that there is always a distinct stand-in for each vertex in the set.

Definition 2 *A* 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. $C = C_u \cup C_m$ (the crown) *is an independent set in G.*
3. $|C_m| = |H|$, *and there is a perfect matching between C_m and H.*

Crown-decomposition is a recently introduced idea that supports nontrivial and powerful preprocessing (reduction) rules for a wide variety of problems, and that performs very well in practical implementations [CFJ03, F03, ACFL04]. It has recently been shown that if a graph admits a crown decomposition, then a crown decomposition can be computed in polynomial time [AS04]. The following theorem can be deduced from [CFJ03, page 7], and [F03, page 8].

Theorem 1 *Any graph G with an independent set I, where $|I| \geq \frac{n}{2}$, has a crown decomposition (H, C, R), where $H \subseteq N(I)$, that can be found in linear time, given I.*

For most problems, including k-K_3-PACKING, it is not clear how a crown decomposition can directly provide useful information. We introduce here the idea of creating an auxiliary graph model where a crown decomposition in the auxiliary graph is used to identify preprocessing reductions for the original graph.

For k-K_3-PACKING we will show that an auxiliary graph model can be created to reduce large independence sets in the problem instance. Consider an independent set I in a graph G. Let E_I be the set of edges that are sponsored by the vertices of I.

The auxiliary model that we consider is a bipartite graph G_I where we have one vertex u_i for every vertex v_i in I and one vertex f_j for every edge e_j in E_I. For simplicity, we let both sets $\{e_j \mid e_j \in E_I\}$ and $\{f_j \mid e_j \in E_I\}$ be denoted by E_I. The edges of G_I are defined as follows: let $u_i f_j$ be an edge in G_I if and only if u_i sponsors f_j.

We now prove the following generalization of Reduction Rule 4.

Reduction Rule 5 *If G_I has a crown decomposition $(H, C_m \cup C_u, R)$ where $H \subseteq E_I$ then G has a k-K_3-packing $\Leftrightarrow G \setminus C_u$ has a k-K_3-packing.*

Proof. Assume on the contrary that G_I has a crown decomposition $(H, C_m \cup C_u, R)$, where $H \subseteq E_I$ and G has a k-K_3-packing W^* but $G \setminus C_u$ has no k-K_3-packing. This implies that some of the vertices of C_u were used in the k-K_3-packing W^* of G.

Let H^* be the set of vertices in H whose corresponding edges in G use vertices from C to form K_3s in the k-K_3-packing W^* of G. Note that vertices in C_u can only form K_3s with edges of G that correspond to vertices in H. Observe that each edge corresponding to a vertex in H^* uses exactly one vertex from C. Further, $|H^*| \leq |H|$. By these two observations it is clear that every edge whose corresponding vertex is in H^* can be assigned a vertex from C_m to form a K_3. Thus C_u is superfluous, contradicting the assumption. □

Observation 1 *If a bipartite graph $G = (V \cup V', E)$ has two crown decompositions (H, C, R) and (H', C', R') where $H \subseteq V$ and $H' \subseteq V$, then G has a crown decomposition $(H'' = H \cup H', C'' = C \cup C', R'' = R \cap R')$.*

It is easy to check that all properties of a crown decomposition hold for (H'', C'', R'').

Lemma 1 *If G has an independent set I such that $|I| > 2|E_I|$ then G is reducible.*

Proof. Assume on the contrary that G has an independent set I such that $|I| > 2|E_I|$ but G is not reducible.

By Theorem 1 the bipartite model G_I as described above has a crown decomposition $(H, C = C_m \cup C_u, R)$ where $H \subseteq N(I)$ and consequently $C \subseteq I$. If $|I \setminus C| > |E_I|$ then $G_I \setminus C$ has a crown decomposition (H', C', R'), where $H' \subset N(I)$. By Observation 1 (H, C, R) and (H', C', R') could be combined to form a bigger crown. Let $(H'', C'' = C''_m \cup C''_u, R'')$ be the largest crown decomposition that can be obtained by repeatedly finding a new crown in $I \setminus C$ and combining it with the existing crown decomposition to form a new head and crown.

By our assumption $C''_u = \emptyset$. Since $|C''_m| = |H''| \leq E_I$ and it follows from Theorem 1 that $|I \setminus C''_m| \leq |E_I|$ (otherwise a new crown could be formed), we have that $|I| = |C''_m| + |I \setminus C''_m| \leq |E_I| + |E_I| \leq 2|E_I|$ contradicting the assumption that $|I| > 2|E_I|$. □

5 Computing a cubic kernel

We now introduce a polynomial time algorithm that either produces a k-K_3-packing or finds a valid reduction of any input graph $G = (V, E)$ of at least a certain size. We show that this algorithm gives an $O(k^3)$ kernel for k-K_3-PACKING.

The algorithm has the following steps:

1. Greedily, find a maximal K_3-packing W in G. If $|V(W)| \geq 3k$ then ACCEPT.
2. Find a maximal matching Q in $G \setminus V(W)$. If a vertex $v \in V(W)$ sponsors more than $3k - 3$ matched edges, then v can be reduced by Reduction Rule 3.

3. If possible, reduce the independent set $I = V \setminus (V(W) \cup V(Q))$ with the technique described in Section 4.

We now give the following lemma to prove our result:

Lemma 2 *If $|V| > 108k^3 - 72k^2 - 18k$ then the preprocessing algorithm will either find a k-K_3-packing or it will reduce $G = (V, E)$.*

Proof. Assume on the contrary to the stated lemma that $|V| > 108k^3 - 72k^2 - 18k$, but that the algorithm produced neither a k-K_3-packing nor a reduction of G.

By the assumption the maximal packing W is of size $|V(W)| < 3k$.

Let Q be the maximal matching obtained by step 2 of the algorithm.

Claim 1 $|V(Q)| \leq 18k^2 - 18k$

Proof. Assume on the contrary that $|V(Q)| > 18k^2 - 18k$. Observe that no edge in $G \setminus V(W)$ can sponsor a vertex in $G \setminus V(W)$ as this would contradict that W is maximal, therefore all edges in the the maximal matching Q are sponsored by at least one vertex in $V(W)$. If $|V(Q)| > 18k^2 - 18k$, Q contains more than $9k^2 - 9k$ edges. Thus at least one vertex $v \in V(W)$ sponsors more than $\frac{9k^2-9k}{3k} = 3k - 3$ edges. Thus v should have been removed by Reduction Rule 3, contradicting the assumption that no reduction of G took place. We have reached a contradiction, thus the assumption that $|V(Q)| > 18k^2 - 18k$ must be wrong. □

Let $I = V \ (V(W) \cup V(Q))$. Note that I is an independent set.

Claim 2 $|I| \leq 108k^3 - 90k^2$

Proof. Assume on the contrary that $|I| > 108k^3 - 90k^2$. Observe that each edge that is sponsored by a vertex of I is either in the subgraph of G induced by $V(W)$, or is an edge between $V(W)$ and $V(Q)$. The are at most $|E_I| = |V(Q)||V(W)| + |V(W)|^2 \leq (18k^2 - 18^k) \cdot 3k + (3k)^2 \leq 54k^3 - 45k^2$ such edges. By Lemma 1 there are no more than $2|E_I| = 108k^3 - 90k^2$ vertices in I, which contradicts the assumption that $|I| > 108k^3 - 90k^2$. □

Thus the total size $|V| = |V(W)| + |V(Q)| + |I| \leq 3k + 18k^2 - 18k + 108k^3 - 90k^2 = 108k^3 - 72k^2 - 18k$. This contradicts the assumption that $|V| > 108k^3 - 72k^2 - 18k$.

□

Corollary 1 *Any instance (G, k) of k-K_3-PACKING can be reduced to a problem kernel of size $O(k^3)$.*

Proof. This follows from Lemma 2, as we can repeatedly run the algorithm until it fails to reduce the graph further. By Lemma 2 the resulting graph is then of size $O(k^3)$. □

Note that a $O(k^3)$ kernel gives us a trivial *FPT*-algorithm by testing all $O(\binom{k^3}{3k})$ subsets in a brute force manner. This leads to an $O(2^{9k \log k} + poly(n, k))$ algorithm. However, we will show in the next section that another *FPT* technique yields a faster algorithm.

6 Winning the *FPT* runtime race

In this section we give a faster *FPT*-algorithm using the technique of "greedy localization" and a bounded search tree.

We begin with the following crucial observation.

Observation 2 *Let W be a maximal K_3-packing, and let W^* be a k-K_3-packing. Then for each K_3 C of W^* we have that $V(C) \cap V(W) \neq \emptyset$.*

Proof. Assume on the contrary that there exists a K_3 C in W^* such that $V(C) \cap V(W) = \emptyset$. This implies that $V(C) \cup V(W)$ is a K_3-packing contradicting that W is a maximal packing. □

Theorem 2 *It is possible to determine whether a graph $G = (V, E)$ has a k-K_3-packing in time $O(2^{2k \log k + 1.869k} n^2)$.*

Proof. Let W be a maximal K_3-packing. If $|V(W)| \geq 3k$ we have a K_3-packing. Otherwise, create a search tree T. At each node we will maintain a collection $S^i = S^i_1, S^i_2, \ldots, S^i_k$ of vertex subsets. These subsets represent the k triangles of the solution, and at the root node all subsets are empty.

From the root node, create a child i for every possible subset W_i of $V(W)$ of size k. Let the collection at each node i contain k singleton sets, each containing a vertex of W_i.

We say that a collection $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a *partial solution* of a k-K_3-packing W^* with k disjoint triangles $W^*_1, W^*_2, \ldots, W^*_k$ if and only if $S^i_j \subseteq V(W^*_j)$ for $1 \leq j \leq k$.

For a child i, consider its collection $S_i = S^i_1, S^i_2, \ldots, S^i_k$. Add vertices to S^i_1 such that S^i_1 induces a K_3 in G, continue in a greedy fashion to add vertices to S^i_2, S^i_3 and so on. If we can complete all k subsets we have a k-K_3 packing. Otherwise, let S^i_j be the set first set which is not possible to complete, and let V' be the vertices we have added to S^i so far. We can now make the following claim.

Claim 3 *If $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a partial solution then there exists a vertex $v \in V'$ such that $S^i = S^i_1, \ldots, (S^i_j \cup \{v\}), \ldots, S^i_k$ is a partial solution.*

Proof. Assume on the contrary that $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a partial solution but that there exists no vertex $v \in V'$ such that $S^i = S^i_1, (S^i_j \cup \{v\}), \ldots, S^i_k$ is a partial solution. This implies that $V(W^*_j) \cap V' = \emptyset$, but then we could add $V(W^*_j) \setminus S^i_j$ to S^i_j to form a new K_3, thus contradicting that it was not possible to complete S^i_j. □

We now create one child u of node i for every vertex in $u \in V'$. The collection at child u is $S^i = S^i_1, (S^i_j \cup \{u\}), \ldots, S^i_k$. This is repeated at each node l, until we are unable to complete any set in node l's collection, i.e. $V' = \emptyset$.

By Observation 2 we know that if there is k-K_3-packing then one of the branchings from the root node will have a partial solution. Claim 1 guarantees that this solution is propagated down the tree until finally completed at level $2k$.

At each level the collections S at the nodes grow in size, thus we can have at most $2k$ levels in the search tree. Observe that at height h in the search tree $|V'| < 2k - h$, thus fan-out at height h is limited to $2k - h$. The total size of the tree is then at most $\binom{3k}{k} 2k \cdot (2k-1) \cdot \ldots = \binom{3k}{k} \cdot 2k! = \frac{(3k)!}{k!}$. Using Stirling's approximation and suppressing some constant factors we have $\frac{(3k)!}{k!} \approx 3.654^k \cdot k^{2k} = 2^{2k \log k + 1.869k}$. At each node we need $O(n^2)$ time to maximize the sets. Hence, the total running time is $O(2^{2k \log k + 1.869k} n^2)$ □

Note that it is, of course, possible to run the search tree algorithm from this section on the kernel obtained in the previous section. The total running time is then $O(2^{2k \log k + 1.869k} k^6 + p(n, k))$. This could be useful if n is much larger than k as the additive exponential (rather than multiplicative) factor becomes significant.

7 Packing arbitrary graphs

In their paper from 1978 Hell and Kirkpatrick [HK78] prove that k-H-packing for any connected graph H of 3 or more vertices is *NP*-complete. We will in this section show that our search tree technique for k-K_3-packing easily generalizes to arbitrary graphs H, thus proving that packing any subgraph is in FPT.

> k-H-PACKING
> INSTANCE: Graph $G = (V, E)$
> PARAMETER: k
> QUESTION: Does G have at least k disjoint copies of H?

Theorem 3 *It is possible to determine whether a graph $G = (V, E)$ has a k-H-packing in time $O(2^{k|H| \log k + 2k|H| \log |H|} n^{|H|})$.*

Proof. The proof is analogous to the proof of Theorem 2. However, as we no longer can depend upon perfect symmetry in H (since H is not necessarily complete), we must maintain a collection of ordered sequences at each tree-node. Each sequence represents a partial H-subgraph.

The possible size of V' increases to $k|H| - k$. Then when we want to determine which v of V' to add to the sequence, we must try every v in every position in H. Thus the fan-out at each node increases to $k|H|^2 - k|H|$. The height of the tree likewise increases to at most $k|H| - k$. Thus the new tree size is $\binom{k|H|}{k}(k|H|^2 - k|H|)^{k|H|-k}$, which is strictly smaller than $k^{k|H|}|H|^{2k|H|}$ or $2^{k|H| \log k + 2k|H| \log |H|}$. □

8 Summary and open problems

Our main results in the two FPT races are:

(1) We have shown an $O(k^3)$ problem kernel for the problem of packing k K_3s. We remark that our approach here does generalize to packing K_ts, for any fixed t, yielding an $O(k^t)$ kernel.

(2) We have shown that for any fixed graph H, the problem of packing k Hs is in FPT with a parameter function of the form $O(2^{O(k \log k)})$.

In addition to "upper bound" improvements to these initial results, which would be the natural course for further research — now that the race(s) are on — it would also be interesting to investigate lower bounds, if possible.

It would be interesting to investigate the "optimality" of the form of our *FPT* results in the sense of [CJ03, DEFPR03]. Can it be shown that there is no $O(2^{o(k)})$ *FPT* algorithm for k-H-PACKING unless $FPT = M[1]$? Can our result be improved to an *FPT* algorithm with a single exponential parameter function of the form $O(2^{O(k)})$?

Many parameterized problems admit linear problem kernels. In fact, it appears that most naturally parameterized problems in *APX* are in *FPT* and have linear problem kernels. However, it seems unlikely that *all FPT* problems admit linear kernels. We feel that PACKING K_tS is a natural candidate for an *FPT* problem where it may not be possible to improve on $O(k^t)$ kernelization. Techniques for the investigation of lower bounds on kernelization are currently lacking, but packing problems may be a good place to start looking for them.

References

[AS04] F. AbuKhzam and H. Suters, Computer Science Department, University of Tennessee, Knoxville, private communications, Dec. 2003.

[ACFL04] F. Abu-Khzam, R. Collins, M. Fellows and M. Langston. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. *Proceedings ALENEX 2004*, Springer-Verlag, *Lecture Notes in Computer Science* (2004), to appear.

[CFJ03] B. Chor, M. Fellows, and D. Juedes. Saving k Colors in Time $O(n^{5/2})$. Manuscript, 2003.

[CJ03] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences* 67 (2003).

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems. Electronic Notes in Theoretical Computer Science 78 (2003), 205–218.

[DF99] R. Downey and M. Fellows. *Parameterized Complexity* Springer-Verlag (1999).

[DFS97] R. Downey, M. Fellows and U. Stege, Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability, in: *Contemporary Trends in Discrete Mathematics*, (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science 49, pages 49-99, 1999.

[E65] J.Edmonds. Paths, trees and flowers, *Can.J.Math.*, 17, 3, pages 449-467, 1965.

[F03] M.Fellows. Blow-ups, Win/Wins and Crown Rules: Some New Directions in *FPT*. *Proceedings WG 2003*, Springer Verlag LNCS 2880, pages 1-12, 2003.

[HK78] P. Hell and D. Kirkpatrick. On the complexity of a generalized matching problem. *Proceedings of 10th ACM Symposium on theory of computing*, pages 309-318, 1978.

[HS89] C. A. J. Hurkens, and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems, *SIAM J. Disc. Math. 2*, pages 68-72, 1989.

[JZC04] W. Jia, C. Zhang and J. Chen. An efficient parameterized algorithm for m-set packing, *Journal of Algorithms*, to appear.

[K91] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete, *Inform. Process. Lett.* 37, pages 27-35, 1991.

[W03] G. Woeginger. Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization - Eureka! You shrink!*, M. Juenger, G. Reinelt and G. Rinaldi (eds.). LNCS 2570, Springer, pages 185-207, 2003.