Algorithm Design under Consideration of Different Resources and a Faulty World

Frank Kammer

Institut für Informatik, Universität Augsburg, Germany
kammer@informatik.uni-augsburg.de

1 Introduction

In 1843, Lovelace [70] already emphasized the importance that Babbage's mechanical analytical engine has a fast running time:

> *One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.*

From this time on, the most studied and most important resource in algorithm design is the running time as, e.g., stated by Beame, Saks, Sun, and Vee [14] or Demaine and López-Ortiz [42]. For most of the problems, a sequential (non-parallel) algorithm has at least a running time linear in the size of the input since it must read the whole input before solving the problem. Thus, a linear running time is the best that we can hope for.

In the last forty years, while searching for algorithms with a small running time, more and more problems are characterized to be NP-hard; for an overview see Garey and Johnson [67]. It is commonly believed that NP-hard problems cannot be solved in polynomial time. However, many real world applications need solutions for such problems. As a consequence, a new field part of the ALGORITHMS AND DATA STRUCTURES research area was initiated about 20 years ago: the so-called *parameterized complexity*. Here, one tries to measure the running time of an algorithm (for an NP-hard problem) by means of several parameters. If there is a parameter k in addition to the size n of the instance such that, for a suitable constant c and a suitable function f, the running time of an algorithm is $O(n^c \cdot f(k))$, then the algorithm (as well as the problem for which the algorithm is designed) is called *fixed parameter tractable* (*FPT*) [46]. Two lines of research appeared: One that tries the function f to grow as slowly as possible and another that tries to obtain a constant c as small as possible. If $c = 1$ and thus the best one can hope for, the algorithm (and the problem) is also called *fixed parameter linear* (*FPL*) [71]. One common approach to develop new parameterized algorithms for graph problems is to use so-called *tree decompositions* and so-called *kernelizations*. A tree decomposition is an auxiliary structure that often allows us to generalize algorithms for trees also to work on general graphs. The running time depends on the 'similarity' of the input graph to a tree, measured by a parameter called *treewidth*. The treewidth, introduced by Robertson and Seymour [109], is one of the basic parameters in graph theory. Many NP-hard problems are solved on graphs G with small treewidth by the following two steps: First, compute a tree decomposition for G of width $k \in \mathbb{N}$ and second, solve the problem by using this tree decomposition. There is a trade-off between the running times of these two steps depending on our choice of k.

A kernelization is a kind of a preprocessing algorithm that transforms a given instance to a new, usually smaller instance of the same problem by applying so-called *reduction rules*. If the new instance satisfies certain properties, it is called a *kernel*. The size of the kernel is a measure of its quality and determines the subsequent running time to solve the problem. Usually, the kernelization is repeated as long as possible, however, similar as it was the case for tree decompositions: there is a trade-off between the size of the kernel and the time of its computation time. The next two sections describe further details to obtain FPL algorithms for the two approaches tree decomposition and kernelization.

Langton [92] stated that time and space are the most common measures for computations. Beame et al. mention in their paper [14] only time and space as important resources for algorithms. One of the earliest papers focusing on computation space is by Stearns, Hartmanis, and Lewis [119] in 1965. If $n \in \mathbb{N}$ is the number of so-called *words* of the input, each algorithm must use $\Omega(n)$ words of memory since the memory contains the input. Moreover, many problems can be solved with $O(n)$ words of memory. At first glance, it seems that it is trivial to design algorithms with the asymptotically smallest memory consumption possible. However, if one wants to distinguish between the memory storing the input, the working memory and the memory storing the output, the situation changes. To develop space-saving algorithms, as it is usually the case, we restrict in this paper the access to the memory: read-only access to the input, a read-write working memory, and write-only access to an output memory. We say that an algorithm *works with s bits* if it can operate correctly with a working memory of that size. There are two typical models of computations, a *Turing machine* (*TM*) that only allows sequential access to its memory and a *random-access model* (*RAM*). Many variations of the RAM are defined. In the *word RAM* [64], the memory is partitioned into *words*, each consisting of $w \in \mathbb{N}$ consecutive bits, and accessing a word as well as executing the usual operations on operators each consisting of a word runs in $O(1)$ time. Some authors consider a RAM where the input can be permuted, but not destroyed [27] or where the input can be modified during the process of answering a query, but it has to be restored to its original state afterwards [33]. Another model used with sequential access used by some authors is the *multi-pass streaming* model [99] where the input is assumed to be held in a read-only sequentially-accessible medium, and the main optimization target is the number of passes an algorithm makes over the input.

To read the input, one needs a counter that allows us to address the words of the input. To implement such a counter, we need at least a logarithmic number of bits with respect to the size of the input. In other words, to read the whole input consisting of $n \in \mathbb{N}$ words we need $\Theta(\log n)$ bits of working space by our restrictions to the input and output memory. Several space-saving results were developed on the Turing machine around forty years ago that works with (poly)logarithmic bits; for an overview see [78]. In particular, using Savitch's famous theorem [115] we can test connectivity in directed n-vertex graphs with only $\Theta(\log^2 n)$ bits of working space, but in a time not polynomial in n. The classical area of finding algorithms for the Turing machine that operate in (poly)logarithmic space is still very active [40, 50]. The Turing machine is a nice and simple model, however, for the analysis of space-saving algorithms that are "in some sense" also time-efficient as well as for algorithms with a small time-space product, the word RAM is a more realistic model of computation.

Let us call an algorithm for a problem *space-efficient* if its space consumption is smaller than the space requirements of the standard algorithms for the problem and if it (almost) matches the time bounds of those algorithms. An overview of space-efficient

algorithms on the word RAM is given in Section 4.

Other resources analyzed by algorithm designers are transmission speed, temporary disk usage, long-term disk usage, power consumption, total cost of ownership, response time to external stimuli, etc. These resources are analyzed comparatively rarely and we do not focus on them here.

So far the goal was to find algorithms with an economical use of resources. To have such an algorithm is often not enough due to several reasons, e.g., since the input data is not precise and thus the solution found by the algorithm can be far away from the best solution for the "real" world. This is in particular the case if the computation executes subtract or division operations. Moreover, if the input consists of non-integer numbers, the computation itself is often not precise. Most of the algorithm designers ignore these problems or hope that the solutions returned by their algorithms are a good approximation of a solution for the real instance. But what properties must the computed result have in order to be considered as acceptable? Schirra [117] points out that even for problems restricted to computational geometry there is no general, widely applicable theory on how to deal with imprecise computation. To handle inexact data some concepts are described in Section 5.

Parts of this paper can be found also in my cited publications.

2 Treewidth

For basic definition in the context of graphs, we refer to the textbook of Diestel [44].

A *tree decomposition* for a graph $G = (V, E)$ is a pair (T, B) with $T = (W, F)$ being a tree and B being a mapping $W \to \{V' \mid V' \subseteq V\}$ such that the following two properties are satisfied:

(1) For each vertex $v \in V$ and for the endpoints of each edge $e \in E$, there is a node $w \in W$ with $B(w)$ contains the vertex and both endpoints, respectively.

(2) For each vertex $v \in V$, the nodes w with $v \in B(w)$ induce a subtree in T.

For each node $w \in W$, $B(w)$ is called the *bag* of w. The *width* of a tree decomposition is the maximal size of a bag minus one. Moreover, the *treewidth* of a graph G is the smallest width of any tree decomposition for G.

Arnborg, Lagergren, and Seese [5] showed that all problems expressible in so-called *linear extended monadic second order logic* (linear EMSO logic) can be solved on n-vertex graphs in a time linear in n and exponential in k if a tree decomposition of width k for the graph is known. Demaine, Fomin, Hajiaghayi, and Thilikos [41] have shown that, for many so-called *bidimensional problems* that are also expressible in linear EMSO logic, one can find a solution of size ℓ in a given n-vertex planar graph—if one exists—in a time linear in n and subexponential in ℓ as described in the next paragraph. Such problems are, e.g., Minimum Dominating Set, Minimum Maximal Matching, and Minimum Vertex Cover, which are all NP-hard on planar graphs.

First, try to find a tree decomposition for G of width $r = \hat{c}\sqrt{\ell}$ for some constant $\hat{c} > 0$. One can choose $\hat{c}$ such that, if the algorithm fails, then there is no solution of size at most ℓ. Otherwise, in a second step, use the tree decomposition obtained to solve the problem by well known algorithms in $O(nc^r) = O(nc^{\hat{c}\sqrt{\ell}})$ time for some constant c. Thus, it is very important to compute a tree decomposition in such a time on planar graphs so that the computation is not the bottleneck when using the approach of Demaine et al. or in a time linear in n when using the approach of Arnborg et al.

Robertson and Seymour [110] presented the first algorithm for the computation of the treewidth and a tree decomposition with a running time polynomial in n and exponential in k where here and in the following n denotes the number of vertices and k the treewidth of the graph under consideration. Afterwards, numerous papers with improved running times were published as, e.g., [4, 25, 91, 112]. Finally, Bodlaender [23] showed that a tree decomposition of smallest width can be found in a time linear in n and exponential in k. However, it is NP-hard to determine the treewidth of a given graph [4], and the running time of Bodlaender's algorithm is practically infeasible already for very small k as shown by Röhrig [113]. For a long time, the most practical algorithm with constant approximation ratio was due to Reed [107]. His algorithm computes a tree decomposition of width $3k+2$ in $O(f(k) \cdot n \log n)$ time for some single exponential function f. More precisely, this width is obtained after slight modifications as observed by Bodlaender [22]. Very recently, Bodlaender, Drange, Dregi, Fomin, Lokshtanov, and Pilipczuk [24] presented the first constant factor approximation algorithm for computing a tree decomposition that is single exponential in k and linear in n.

The algorithm achieving the so far smallest approximation ratio of the treewidth for general graphs among the algorithms with a running time polynomial in n and k is the algorithm of Feige, Hajiaghayi, and Lee [59]. It constructs a tree decomposition of width $O(k\sqrt{\log k})$ thereby improving the bound $O(k \log k)$ of Amir [3]. In particular, no algorithms with constant approximation ratios are known for general graphs that are polynomial in both, the number of vertices and the treewidth.

Better constant-factor approximation algorithms are known for the special case of planar graphs, but it is still an open question if the treewidth of a planar graph can be determined in polynomial time or if this problem is NP-hard. Seymour and Thomas [118] showed that the so-called *branchwidth* $\mathrm{bw}(G)$ and a so-called *branch decomposition* of minimal width can be computed for a planar graph G in $O(n^2)$ and $O(n^4)$ time, respectively. For each graph G, its branchwidth $\mathrm{bw}(G)$ is closely related to its treewidth $\mathrm{tw}(G)$; in detail, $\mathrm{bw}(G) \leq \mathrm{tw}(G)+1 \leq \max(3/2\,\mathrm{bw}(G), 2)$ [111]. A branch decomposition of a graph G can be used directly—like a tree decomposition—to support efficient algorithms, but can also be turned into a tree decomposition of width linear in the width of the branch decomposition in a time linear to the size of the given graph plus the size of the branch decomposition [111]. Gu and Tamaki [72] improved the running time to $O(n^3)$ for constructing a branch decomposition and thus for finding a tree decomposition of width $O(\mathrm{tw}(G))$. They also showed that one can compute a tree decomposition of width $(1.5+c)\mathrm{tw}(G)$ for a planar graph G in $O(n^{(c+1)/c} \log n)$ time for each $c \geq 1$ [73].

Kammer and Tholey [84] presented the first algorithm that finds a tree decomposition of width $O(k)$ in a time linear in n and subexponential in k. In more detail, the algorithm has a running time of $O(nk^3 \log k)$, i.e., given a planar graph, one can determine a constant factor approximation of its treewidth in a time linear in n and polynomial in k. In contrast to general graphs, on planar graphs many graph parameters as branchwidth and rank-width differ only by a constant factor from the treewidth [111, 63, 103]. Thus, the algorithm also computes a constant factor approximation for these parameters on n-vertex planar graphs in a time linear in n. Two years later, Gu and Xu [74] presented an $O(n(\log n)^4 \log k)$-time algorithm with a better approximation ratio for the treewidth of a planar graph.

Kammer and Tholey [86] improved the running time of their algorithm mentioned in the last paragraph by a factor k and generalized their result to *weighted planar graphs* (G, c), i.e., graphs $G = (V, E)$ with a weight function $c : V \to \mathbb{N}$ ($\mathbb{N} = \{1, 2, 3, \ldots\}$).

Roughly speaking, the weighted treewidth of a weighted graph is defined analogously as the (unweighted) treewidth, but instead of counting the vertices of a bag, the weight of the vertices in the bag is summed up. All results shown for weighted graphs and weighted treewidth can be easily applied to unweighted graphs and unweighted treewidth by setting $c(v) = 1$ for all vertices v of G. In detail, Kammer and Tholey's algorithm [86] computes a tree decomposition for a given weighted planar graph in time $O(nk^2 c_{\max} \log k)$ such that the vertices in each bag have a total weight of $(15 + \epsilon)k + 14c_{\max} + O(1) = O(k)$ where $c_{\max}$ denotes the maximum weight over all vertices. This means that a tree decomposition for unweighted graphs of width $(15+\epsilon)k+O(1) = O(k)$ is obtained in $O(nk^2 \log k)$ time, which is a better running time than the one of Gu and Xu [74] for all n-vertex planar graphs of treewidth $k = O(\log n)$. Graphs with a larger treewidth are usually of minor interest since for such graphs it is not clear whether one can efficiently solve the second step of the approach of Demaine et al. described above.

Interestingly, the generalization from unweighted to weighted graphs is possible without increasing the asymptotic running time. Instead, Kammer and Tholey slightly modified their algorithm to bound the number of so-called $(\mathcal{S}, \varphi)$-*components*, which improves the running time by a factor k compared to the running time of the previous published version. Additionally, the usage of weighted graphs simplifies the computation of a tree decomposition on so-called ℓ-*outerplanar graphs* by reduction to 1-*outerplanar graphs*. Another application of weighted treewidth is that one can triangulate graphs with only a little increase in the weighted treewidth, which improves the approximation ratio of the algorithm on unweighted graphs in [86] compared to [84] by about a factor of 4. We want to remark that tree decompositions can be used also on special graph classes [83] and to find approximation algorithms [85].

Since a tree decomposition can be of size $\Theta(nk)$, it is an interesting open problem if the computation of a tree decomposition can be improved by a factor k or to show a non-trivial lower bound on the computation of a tree decomposition. Moreover, many algorithms could be generalized from planar graphs to so-called *bounded genus graphs* or H-*minor free graphs* for a fixed graph H; is such a generalization also possible for the algorithm above computing a tree decomposition on planar graphs?

3 Linear-Time Kernelization

To solve an NP-hard decision problem P, a common approach is to consider P as a *parameterized problem* identified with a language $L \subseteq \Sigma^* \times \mathbb{N}$. An instance I of P is specified by a tuple $(x, k) \in \Sigma^* \times \mathbb{N}$ composed of the problem specification x and a problem-specific *parameter* k. The instance I is *solvable* exactly if the instance is in L.

A *kernelization* for a parameterized problem P, introduced by Downey and Fellows [46], is an algorithm that takes an instance $I = (x, k)$ and maps it in time polynomial in $|x|$ and k to an instance $I' = (x', k')$ such that the size of x' is bounded by a computable function f in k, k' is bounded by a function g in k, and the following condition holds: I is solvable if and only if I' is solvable. Then, I' is called a (*polynomial-sized*) *kernel* for I (if f is a polynomial). It is a well-known fact that a kernelization for a decision problem in NP yields an FPT algorithm for the problem: First run the kernelization, then run a best known exponential-time algorithm on the kernel obtained, and finally output the answer of the algorithm, which is either yes or no.

For numerous problems, a race has started to obtain better and better bounds on the size of the kernel and, thus, on the corresponding running times with more

and more slowly growing functions f. For a list of kernelization results for many important problems, each with the currently smallest kernel, see the Table-of-FPT-races webpage [61]. In contrast, the only linear-time kernelization known to the author are the following: Vertex Cover[46, 47] and (Bi)cluster Graph Editing [105] on general undirected graphs, d-Hitting Set [19] on undirected hypergraphs, Feedback Vertex Set [87], Face Cover [87], and Dominating Set [20, 76] on undirected planar graphs, and Rooted k-Leaf Outbranching Problem [80] on digraphs. If polynomial time instead of only linear time is allowed, better bounds on the obtained kernel can often be obtained as shown below.

problem	polynomial time	linear time
Vertex Cover	$O(k)$ vertices [34, 101]	$O(k^2)$ [46, 47]
Cluster Graph Editing	$O(k)$ [60, 75]	$O(k^4)$ [105]
d-Hitting Set	$O(k^{d-1})$ vertices [19]	$O(k^d)$ [19]
Rooted k-Leaf Outbranching	$O(k^2)$ [38]	$O(k^7)$ [80]

Note that by the approach described above a linear-time kernelization for decision problems in NP yields an FPL algorithm. Interestingly, for all the problems mentioned in the previous paragraph, the following approach can be used to compute not only a yes-no answer, but also a solution in the time of an FPL algorithm. Given an instance $I = (x, k)$ of such a problem P, we first compute a polynomial-sized kernel I' of I using one of the linear-time kernelizations for P cited above. Since the size of I' is $poly(k)$, we next solve I' by the best known exponential-time algorithm in a time $f(k)$ for some function f. Finally, we transform the solution found for I' into a solution for I in linear time. (Unfortunately, the definition of kernelization does not require that this transformation can be done efficiently or even in linear time. Often, it is possible, but one independently has to check this for each kernelization.) Consequently, a solution can be found in $O(n + f(k))$ time where $n = |x|$, i.e., we have an *FPL optimization algorithm* for P.

A reason why there are only few linear-time kernelizations is that, for an instance of size n, the running time of the computation of a kernel can typically be bounded only by a polynomial because of the following two reasons: one usually spends $\Omega(n)$ time to find a 'place' where to apply a reduction rule, and cascading effects make it necessary to search again and again the whole instance for another application of a reduction rule. In addition, reduction rules are usually applied in an exhaustive way so that one can bound the number of applications of a reduction rule only by $\Omega(n)$.

In contrast, to find a linear-time kernelization for Vertex Cover, it is enough to iterate over all vertices and to remove the vertices of large degree. For (Bi)Cluster Graph Editing, the graph is divided by a so-called *modular decomposition* into small subgraphs such that one can independently shrink each subgraph in a time linear to the size of the subgraph. For d-Hitting Set, the main approach is to iterate over all edges twice while searching for some local structure, a so-called *sunflower*. To obtain a linear-time kernelization for Dominating Set and for Face Cover on planar graphs, one exploits the fact that the faces of a planar embedding of each graph divide the graph into several subgraphs, where each consists of the vertices and edges on the boundary of a face. For Face Cover, after making the graph biconnected, one can handle the graph on the boundary of each face independently. For Dominating Set, instead

of considering single faces, one considers a set of few or similar faces. For Rooted k-Leaf Outbranching Problem, a breadth-first-search tree is used to divide the given digraph into paths and edges between these paths.

To sum up, all linear-time kernelizations mentioned above divide the given graph into small subgraphs such that the reduction rules can be applied efficiently in each subgraph G' and such that the application of a reduction rule in another subgraph G'' does not make it necessary to consider G' again, i.e., cascading effects are avoided. For some problems, the division is implicit (each vertex is its own part), for other problems, the division is given by some natural properties (planar embedding) or can be computed explicitly (modular decomposition, breadth-first-search tree).

As an open problem, there are several known kernelization results that can be re-engineered to (hopefully) obtain a linear running time. Usually, new ideas are necessary to obtain a linear running time:

- Van Bevern et al. [20] used old reduction rules, but in contrast to earlier papers, special criteria had to be found when to stop applying the reduction rules.
- Kammer [80] applied an old reduction rule 'stepwise'.

4 Space-Efficient Algorithms

Even though computer memories become larger and cheaper every year, the wish to store data increases even "faster"; e.g., every day, about 65.000 clips are added to YouTube [108]. The World Data Center Climate [122] stores several petabyte of climate data and a large part of the data is accessible via the internet. Of course, we can not modify that data. With a nowadays internet connection for private persons (e.g., 100 MBit) the largest currently available disk (8 TB) is already full after a few days. In other words, we can access many data very fast, but not make a private copy. In addition, we want to use tiniest devices that can communicate with the internet, but have only a small amount of working memory. Standard actions like storing many pictures or songs decreases the available memory even more. In practice, there are already several navigation programs that work only online; one important reason for this is to reduce the space consumption of the devices.

Moreover, new storage devices, called solid-state disks, allow us an unlimited number of readings, but only a relatively small number of write operations until lifetime ends. Thus, it is an advantage if we can run the computation without write operations to the solid-state disks, i.e., run it within the internal memory. There can be also security or technical reasons (e.g., a DVD) that forbid to write the input. Furthermore, if several algorithms run on the same input, the algorithms should also avoid modifying the input. Motivated by these and many other applications, several space-efficient algorithms are published.

One of the first problems considered for space-efficient algorithms is sorting [99]. In a general sequential model of computation, Borodin and Cook [26] showed the lower bound $\Omega(n^2)$ for the time-space product of the *ranking problem* of n elements, i.e., the problem to produce a list in arbitrary order consisting of a tuple (x, y) for each given element x with y being the rank of x. Afterwards, Beame [13] showed the same bound for sorting n elements in the so-called *branching-program model*. Pagter and Rauhe [104] described an algorithm for sorting n elements that works with $s \geq \log n$ bits and runs in $O(n^2/s + n \log s)$ time. Other researchers [30, 32, 64, 100, 106] have considered a problem related to sorting, the so-called *selection problem* where the goal

is, given a set of elements and a number $k \in \mathbb{N}$, to determine the kth smallest number among the set. Elmasry, Juhl, Katajainen, and Satti [52] presented an algorithm to solve the selection problem with a running time $O(n \log^*(n/s) + n(\log n)/\log s)$ using s bits where $(\log n)^3 \leq s \leq n$.

Also various space-efficient algorithms are published for problems in computational geometry. Asano, Mulzer, Rote, and Wang [8] showed how to solve many geometric problems (e.g., triangulating a planar point set, Delaunay triangulation, and Voronoi diagram) in $O(n^2)$ time and $O(\log n)$ bits of working space where n denotes the number of given points. Chan and Chen [31] presented a randomized algorithm for linear programming that, given an array of n half-spaces in a constant number of dimensions, computes the lowest point in their intersection in $O(n)$ expected time and works with $O((\log n)^2)$ bits. In addition, they described a randomized algorithm for computing the convex hull of n points sorted from left to right in the plane (i.e., in two dimensions) that works with $O(n^\epsilon)$ bits and runs in $O(n/\epsilon)$ expected time for any fixed $\epsilon > 0$.

Later, several papers with time-space trade-offs were published. Darwish and Elmasry [39] solved the convex-hull problem to optimality with an algorithm that works with $\Theta(s)$ bits ($s \geq \log n$) and runs in $O(n^2/s + n \log s)$ time. An algorithm for computing a convex hull of a simple polygon was presented by Barba, Korman, Langerman, Silveira, and Sadakane [11]. They developed a general framework that can be applied to incremental linear-time algorithms that, given n objects, use a stack of size $O(n \log n)$ bits and possibly $O(\log n)$ further bits. The framework allows us to reduce the space consumption of the algorithm to either $O(s)$ bits ($\log n \leq s \leq (\log n)^2$) at a price of an increased running time of $O(n^2/2^{s/\log n})$ or to $O(p(\log n)^2/\log p)$ bits for any $2 \leq p \leq n$ and time $O(n(\log n)/\log p)$. The framework can be used for computing the convex hull of a simple polygon as well as a triangulation of a monotone polygon, the shortest path between two given points inside a monotone polygon, and the visibility profile of a point inside a simple polygon. Furthermore, Barba, Korman, Langerman, and Silveira [10] described an algorithm for computing the visibility of a simple polygon with n vertices that works with only $O(\log n)$ bits and has a running time of $O(nr) = O(n^2)$ where r is the number of the so-called *reflex vertices* of the polygon that are part of the output.

The following data structure was presented by Asano, Buchin, Buchin, Korman, Mulzer, Rote, and Schulz [6]. Given a simple polygon P by the ordered sequence of its $n \in \mathbb{N}$ vertices and an integer $s \in \{\log n, \ldots, n \log n\}$ that specifies the used working space $O(s)$, after $O(n^2)$ preprocessing time, the edges of a shortest path between any two points inside P can be executed in $O(n^2(\log n)/s)$ time. To obtain the result, Asano et al. also show, given an n-vertex plane graph with a straight-line embedding, how to triangulate it in $O(n^2)$ time and $O(\log n)$ bits of working space. For the latter result, a time-space trade-off was shown by Korman, Mulzer, van Renssen, Roeloffzen, Seiferth, and Stein [89]: Given a set of n points and an integer $s \in \{\log n, \ldots, n \log n\}$, they described an algorithm for computing a triangulation that works with $O(s)$ bits and runs in $O((n^2/s)(\log n) + n(\log n) \log s)$ time as well as a randomized algorithm for finding the Voronoi diagram that works with the same number of bits and runs in $O((n^2/s)(\log n)(\log s) + n(\log s) \log^* s)$ expected time.

Many geometric problems are usually solved with plane-sweep algorithms [16, 17, 36]. Typically, a plane-sweep algorithm uses a priority queue to generate the upcoming events and a balanced binary search tree to store and query the objects that cross the sweep line. Since $\Theta(n)$ objects might be part of the search tree, a standard plane-sweep algorithm can need $\Theta(n \log n)$ bits. Chan [29] showed that the closest-pair problem where all of the n given points have integer coordinates can be solved with a comparison

based sorting algorithm, i.e., with $s \geq \log n$ bits and in $O(n^2/s+n \log s)$ time. Konagaya and Asano [88] gave an algorithm for reporting the intersections of n line-segments that works with $O(s)$ bits and runs in $O((n^2/\sqrt{s})\sqrt{\log n}+k)$ time, where $\log n \leq s \leq n \log n$ and k is the number of intersecting pairs. Afterwards, Elmasry and Kammer [53] presented three techniques that can make plane-sweep algorithms more space-efficient. The *stretching technique* enables us stretch the range of the workspace where a plane-sweep algorithm works by modifying the algorithm such that it considers and stores only some of the objects crossing the sweep line at a time (in return, one has to run several plane-sweep algorithms). The *batching technique* makes it possible to reduce the space needed to represent $\Theta(n)$ objects crossing the sweep line without changing for the worse the running time. The *multi-scanning technique* allows us to divide the plane into a grid of cells and to partition the events in a way "suitable" for running plane-sweep algorithms in the cells. Elmasry and Kammer used their techniques to present several algorithms that all work with $O(s)$ bits for any $s \geq \log n$. In detail, they described an algorithm for counting and enumerating the line-segments intersections that runs in $O((n^2/s^{2/3}) \log s)$ and $O((n^2/s^{2/3})(\log s) + k)$ time, respectively. For the special case when the line segments are horizontal or vertical, they gave counting and enumerating algorithms whose running times is $O((n^2/s)(\log s)^{4/3} + n^{4/3} \cdot (\log n)^{1/3})$ and $O((n^2/s)(\log s)(\log \log s) + n(\log s) + k)$, respectively. In addition, they showed an $O((n^2/s + n \log s) \cdot \sqrt{(n/s) \log n})$-time algorithm to calculate Klee's measure of axis-parallel rectangles as well as a space-efficient algorithms for closest pair. Given n points with arbitrary coordinates, their closest-pair algorithm runs in $O(n^2/s + n \log s)$ time. A lower bound of $\Omega(n^{2-\epsilon})$ was given by Yao [123] for the time-space product of the so-called *element-distinctness problem* where ϵ is an arbitrarily small positive constant. This lower bound applies for the closest-pair problem indicating that Elmasry and Kammer's algorithm for closest pair is close to optimal.

In the last two years, researchers started to consider also space-efficient graph algorithms. In contrast to sorting and geometric problems, most of the polynomial-time graph algorithms on n vertex graphs seem to need almost $\Theta(n)$ bits of working space since already the polynomial-time algorithms for the fundamental *directed s-t connectivity problem*, i.e., the problem to find a path with endpoints s and t in a given directed graph, needs almost $\Theta(n)$ bits. More exactly, Barnes, Buss, Ruzzo, and Schieber [12] gave an algorithm for the directed s-t connectivity problem that needs $n/2^{O(\sqrt{\log n})}$ bits of memory when required to run in polynomial time, and a nearly matching lower bound for the so-called *NNJAG model* was proved by Edmonds, Poon, and Achlioptas [49]. For the rest of this section, let n and m denote the number of vertices and edges, respectively, of the given graph. For a definition of the algorithms and further terminology used below, we refer to the textbook of Cormen, Leiserson, Rivest, and Stein [36] and to Kammer and Täubig [82].

Elmasry, Hagerup, and Kammer [51] presented several basic graph algorithms: They showed that a depth-first search (DFS) can be carried out in $O((n + m) \log n)$ time with $((\log_2 3) + \epsilon)n$ bits for arbitrary fixed $\epsilon > 0$. A very similar result was found independently by Asano, Izumi, Kiyomi, Konagaya, Ono, Otachi, Schweitzer, Tarui, and Uehara [7], who need cn bits for an unspecified constant $c > 2$, or $\Theta(mn)$ time. Moreover, Elmasry et al. relaxed the space bound to $O(n)$ bits at the price of an increased running time of $O((n+m) \log \log n)$. They also showed how to achieve a linear running time with $O(n \log \log n)$ bits of working space and how to interpolate between the two latter results with the same time-space product. In addition, they showed how to run a DFS in reverse with only a small penalty of $O(n \log \log n)$ extra bits. Con-

sequently, topological sortings and strongly connected components can be computed in linear time with $O(n \log\log n)$ bits. Although the connected components of a given undirected graph are usually computed by means of DFS, Elmasry et al. observed that this bottleneck can be avoided and showed how to output the connected components in $O(n+m)$ time with $O(n)$ bits and how to compute a shortest path forest—and thus some variant of a breadth-first search (BFS)—within the same resource bounds. Work in progress shows that many other graph problems have space-efficient algorithms.

Very recently, Hagerup and Kammer [77] showed that the constant hidden in the O-notation for the two latter problems can be chosen to be very small without increasing the running time. In detail, the connected components can be computed with at most $(1+\epsilon)n$ bits for every fixed $\epsilon > 0$. Moreover, a shortest path forest can be computed with $(\log_2 3)n + o(n)$ bits. To obtain so tiny constants, they develop a special kind of dictionary, a *choice dictionary*, that seems to be fundamental in space-efficient computing. A choice dictionary maintains a set $\mathcal{S}$ subset of a universe $\mathcal{U}$ of size $n \in \mathbb{N}$ under the usual operations insert, delete, and membership as well as a new operation *choice* that returns an arbitrary element of S. To make the choice dictionary even more useful, Hagerup and Kammer extend the choice dictionary with one set $\mathcal{S}$ and its *basic operations* from above by maintaining $c \in \mathbb{N}$ pairwise-disjoint sets $\mathcal{S}_0, \ldots, \mathcal{S}_{c-1} \subseteq \mathcal{U}$ and new operations, e.g.:

robust iteration: After calling an initialization operation *init*(j) with $j \in \{0, \ldots, c-1\}$, one can iterate over the elements of $\mathcal{S}_j$ by calls of an operation *next*(j) while changes to the client set $\mathcal{S}_j$ through insertions and deletions can be tolerated.

bijection from $\mathcal{S}_j$ to $\{1, \ldots, |\mathcal{S}_j|\}$ for any $j \in \{0, \ldots, c-1\}$: An operation *p-rank*(j) realizes a bijection from $\mathcal{S}_j$ to $\{1, \ldots, |\mathcal{S}_j|\}$ and an operation *p-select*(j) realizes the inverse bijection. The bijections may change after a sequence of insert and delete operations even if before and after the sequence, we have the same set $\mathcal{S}_j$.

As shown by Hagerup and Kammer, there is a choice dictionary for a universe $\mathcal{U}$ of size n on a word RAM with a word length of $w = \Omega(\log n)$ bits that stores one set $\mathcal{S} \subseteq \mathcal{U}$ and supports all its basic operations in constant time and runs with $n + O(n/(\log n)^t)$ bits for any fixed $t \in \mathbb{N}$. Furthermore, all operations mentioned above can be supported on $c \in \mathbb{N}$ sets $\mathcal{S}_0, \ldots, \mathcal{S}_{c-1} \subseteq \mathcal{U}$ in constant time with $n(\log_2 c) + \Theta(n(\log\log n)/(t \log n) + n^\epsilon)$ bits for any fixed $t \in \mathbb{N}$ and fixed $\epsilon > 0$. Note that the latter two space bounds are only $o(n)$ larger than the *information-theoretic lower bound*, i.e., the space needed to store the pure information of the maintained sets (without any auxiliary data) as compactly as possible.

As future work, Hagerup and Kammer started to extend the choice dictionary, which maintains a set $\mathcal{S}$ subset of a universe $\mathcal{U}$, by constant-time operations that allows us to read and write *extra information* with each element in $\mathcal{S}$ such that the total space consumption is $O(|\mathcal{U}| + b)$ where b is the total number of bits stored in the extra information. The usefulness of such an extended choice dictionary was already shown in [51].

5 Faulty World

We now discuss some ideas that can be used to find solutions of instances from a faulty world. However, most of the research described in this section is still in its infancy and is a good starting point for future work.

5.1 Uncertainty

About a hundred years ago, Łukasiewicz and Tarski [95] observed that the world is not so perfect as that it can be modeled with values true and false. As a consequence, they started to generalize Boolean logic to three-valued and later to arbitrary-valued as well as to infinitely-valued logic. In 1965, the infinitely-valued logic was reintroduced by Zadeh [124] as *fuzzy logic* where the truth values are variables that may be any real number between 0 and 1. The real numbers allows us to handle the concept of partial truth. Several formal systems based on fuzzy logic have been discussed since that time and many applications have been proposed, e.g., in linguistics and artificial intelligence. Another attempt to model the world is the *probabilistic logic* [102], i.e., a modification of Boolean logic where the truth values are replaced by probabilities and the results are probabilistic expressions. Unfortunately, there is no efficient algorithm for fuzzy or probabilistic logic since already the check if a clause in propositional logic can be made true is not efficiently solvable—more exactly, the so-called *satisfiability problem* (SAT) is NP-hard [35].

Whereas standard optimization problems are formulated with known parameters, real world problems usually include some parameters that are known only within certain bounds. To deal with such an input, a good approach is to use *robust optimization* to find a *robust solution*, i.e., to find a solution that is feasible for all possible realizations within the uncertainty bounds of the given input and optimal in some sense. For a survey on robust optimization and a long list of its applications, see Gabrel, Murat, and Thiele [66] as well as Beyer and Sendhoff [21]. However, complete protection against all possible realizations often comes at the expense of a very bad solution. Thus, there is also a large branch of the robust-optimization area that searches for a solution guaranteeing only feasibility for most of the possible realizations and so allows us to find a solution whose "distance to optimality" is not too large [15, 18]. One often used approach to find a robust solution is the so-called *minimax regret criterion*, i.e., the minimization of the regret that is highest when one decision has been made instead of another [1, 93]. Moreover, there are also approaches of robust optimization that try to combine robust and stochastic optimization [62].

To deal with the uncertainty, some problems allow us to find a good upper and lower bound for the solutions. Löffler and van Kreveld [94] presented polynomial-time algorithms for the problem to compute the smallest and largest convex hull of a set of points in the plane where only a non-intersecting area for each point is known, but not the exact position. Dorrigiv, Fraser, He, Kamali, Kawamura López-Ortiz, and Seco [45] considered the problem of computing the minimum and maximum cost of a Euclidean spanning tree for input points that are specified via uncertainty areas being disjoint disks. They showed the NP-hardness of the problem and presented approximation algorithms whose ratio depends on a separability parameter of the given instance.

Many optimization problems can be viewed as the problem of selecting a minimum-weight set among all feasible sets where it is often the case that the exact weight of an element is initially only approximately known, but it is possible to obtain the exact weight at an extra cost. Therefore, Erlebach, Hoffmann, and Kammer [57] considered the following *cheapest set problem*. Given a set E of elements, each with an exact weight for which, however, initially only an open uncertainty area of the weight is known, and a collection $\mathcal{S}$ of weighted subsets, find a cheapest set $S \in \mathcal{S}$ with a minimum number of queries that reveals the exact weight of S. Erlebach et al. showed that the cheapest set problem can be solved with at most $d \cdot \mathrm{OPT} + d$ queries, where d is the maximum

cardinality of the given sets in $\mathcal{S}$ and OPT is the optimal number of queries needed to identify a cheapest set in the given instance. They also gave algorithms for several special cases, e.g., where the feasible sets are the bases of a matroid. In addition, they presented matching lower bounds for each of their algorithms.

5.2 Modifications of the Input

Many algorithms in computational geometry only work if the given points or the endpoints/corners of the given objects are in *general position*, i.e., no three points lie on the same line. This assumption is often satisfied for almost all points of a given real-world instance and a few missing points usually do not or only slightly change the solution for many problems as, e.g., convex hull or closest pair. Therefore, Froese, Kanj, Nichterlein, and Niedermeier [65] considered the problem to find a largest subset of a given set of points in general position. They showed that the problem is NP-hard, but FPT for several parameters.

When terrains are used for studies on land erosion, landscape evolution, etc., then the input data is often given as an *imprecise terrain*, i.e., a planar triangulation with an additional interval of possible heights associated with each vertex. However, to use a terrain for computations, it becomes much easier if one can use a *valid realization* of the imprecise terrain, i.e., each vertex has a fixed height that is within its height interval. But what is a good valid realization? It is generally accepted that a terrain with many pits or peaks does not represent a real terrain in a good way and can create problems in the case of pits because water accumulates at them and thus affects the simulations of the water flow. Gray, Kammer, Löffler, and Silveira [69] showed that given an imprecise terrain T with n vertices a valid realization of T with a minimum number of local minima (or local maxima) can be found in $O(n \log n)$ time. They also showed that minimizing the total number of local extrema of T can not be approximated within a factor $O(\log \log n)$ unless P = NP.

5.3 Temporal Graphs

If *networks*, i.e., graphs with edge costs, are used to model communications, connections in social networks, etc., the graphs are not static and change over time. Such graphs have been studied in many contexts as so-called *faulty networks*, *dynamic networks*, and *time-varying networks* [28, 97, 116].

Many papers studied distributed computing in dynamic networks [9, 48, 90]. However, there are not many papers that present algorithms for temporal versions of classical graph problems. Michail and Spirakis [98] considered temporal analogues of traveling salesman problems, Maximum Matching, Path Packing, and Minimum Cycle Cover. They also showed that already the simple problem to find out if a given undirected graph can be explored—exploration is defined precisely in the next paragraph—is NP-hard if the edge set of the graph changes over a discrete time. Such a graph is a time-varying network without edges costs and is called a *temporal graph* $\mathcal{G}$ that consists of a sequence of undirected graphs $G_0 = (V, E_0)$, $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, ..., $G_L = (V, E_L)$ where $L \in \mathbb{N}$ is the *lifetime* of the graph, G_i $(0 \leq i \leq L)$ is the graph in *time step* i, and $G = (V, E_0 \cup \ldots \cup E_L)$ is called the *underlying graph* of $\mathcal{G}$. For the rest of this section, n denotes the number of vertices of the given temporal graph.

A *temporal walk* for $\mathcal{G}$ is an alternating sequence of vertices and edge-time pairs $v_0, (e_0, i_0), v_1, \ldots, (e_{k-1}, i_{k-1}), v_k$ such that $e_j = \{v_j, v_{j+1}\} \in E_{i_j}$ for $0 \leq j \leq k-1$ and $i_0 < i_1 < \cdots < i_{k-1}$. We say that $v_0, \ldots, v_k$ are *visited* by the walk and that the walk

uses $i_{k-1} + 1$ *time steps.* An *exploration schedule* of a temporal graph is a temporal walk that starts at a given vertex s at time 0 and visits all vertices. If a temporal graph has an exploration schedule, we say that the graph can be *explored.* The *temporal graph exploration problem* (TEXP) is the problem of computing a exploration schedule for a temporal graph that uses as few as possible time steps to visit all vertices, and a ρ-approximation algorithm for TEXP is an algorithm that runs in polynomial time and outputs an exploration schedule whose number of used time steps is bounded by ρ times the number of used time steps of an optimal exploration schedule.

To overcome the NP-hardness of TEXP, Michail and Spirakis also considered temporal graphs that are connected in every time step and showed that those temporal graphs can be explored by using $O(n^2)$ time steps, which is an $O(n)$ approximation since each exploration needs at least n time steps. Erlebach, Hoffmann, and Kammer [55] showed that it is NP-hard to approximate TEXP with ratio $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$ even if the graphs are connected in every time step. This motivated them to consider TEXP under the assumption that the underlying graph belongs to a specific graph class. In detail, they showed that temporal graphs that are connected in every step can be explored in $O(n^{1.5}k^2 \log n)$ steps if the underlying graph has treewidth k, in $O(n(\log n)^3)$ steps if the underlying graph is a $2 \times n$ grid, and in $O(n)$ steps if the underlying graph is a cycle or a cycle with a chord.

In many applications, there is some regularity in the presence of edges or a good lower bound on the probability of the presence of the edges is known. Therefore, Erlebach, Hoffmann, and Kammer [56] also focused on general m-edge temporal graphs where each edge of the underlying graph—independent of the other edges—is present with a certain regularity and with a certain probability. Such temporal graphs can be explored in $O(m)$ and $O(m \log n)$ (the latter holds with high probability) time steps, respectively. This means that we have a good schedule for exploring such temporal graphs if the underlying graph is sparse.

5.4 Fault-Tolerant Sensor Networks

Sensor cover problems have been studied in several variants. For an overview see Thai, Wang, Hongwei Du, and Jia [120]. Such studies include *sensor-network problems* where the input consists of a discrete set of points that need to be covered by sensors. For example, Dhawan, Vu, Zelikovsky, Li, and Prasad [43] studied a sensor-network problem where the sensor nodes can adjust their sensing range and proposed a greedy algorithm for the minimum cost sensor cover problem that yields a logarithmic approximation ratio. Zhao and Gurusamy [125] focused on the sensor-network problem with the additional requirement that the sensors that are active at any time are connected. They obtained an algorithm also with logarithmic approximation ratio. Sanders and Schieferdecker [114] showed that the sensor-network problem for sensors represented by unit disks with the objective of lifetime maximization is NP-hard. They also provided a $(1 + \epsilon)$-approximation algorithm using resource augmentation, i.e., their algorithm needs to increase the sensing range of every sensor node by a factor of $1 + \delta$, for some fixed $\delta > 0$. Another kind of modification of the sensor-network problem was done by Kammer, Löffler, Mutser, and Staals [81]. They considered the problem placing towers to cover a terrain. Because of the impreciseness of the terrain that is used for the computation, a correct solution for the imprecise terrain does not cover the whole real terrain. Therefore, Kammer et al. gave up the requirement to cover the whole terrain and presented a simple polynomial time algorithm to compute a set of towers that guards most of the terrain and experimentally evaluated their algorithm on coarse

and fine approximations of real terrains.

A special case of the minimum cost sensor-network problem is the weighted geometric set cover problem where a set of points and a set of weighted unit disks is given and the goal is to compute a cheapest set of disks that covers all points. Many researchers [2, 79, 37, 58, 126] considered this problem because it includes the weighted dominating set problem for unit disk graphs, which is relevant for so-called *routing backbones*.

However, only few papers consider sensor-network problems in combination with fault-tolerance requirements. Vu, Beyah, and Li [121] as well as Marta, Yang, and Cardei [96] presented heuristics for fault-tolerant sensor-network problems. Contrary to their work, Erlebach, Grant, and Kammer [54] gave a $(6 + \epsilon)$-approximation algorithms with provable performance guarantee for a fault-tolerant sensor-network problem. Using that approximation algorithm as a subroutine in the Garg-Könemann algorithm [68], one also obtains the same approximation ratio $6 + \epsilon$ for the problem to maximize the time that a sensor network can run with sensors of limited lifetime.

References

[1] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *Eur. J. Oper. Res.*, 197(2):427–438, 2009. doi:10.1016/j.ejor.2008.09.012.

[2] C. Ambühl, T. Erlebach, M. Mihal'ák, and M. Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Proc. 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006) and 10th International Workshop on Randomization and Computation (RANDOM 2006)*, volume 4110 of *LNCS*, pages 3–14. Springer, 2006. doi:10.1007/11830924_3.

[3] Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010. doi:10.1007/s00453-008-9180-4.

[4] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8(2):277–284, 1987. doi:10.1137/0608024.

[5] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

[6] Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Reprint of: Memory-constrained algorithms for simple polygons. *Comput. Geom. Theory Appl.*, 47(3, Part B):469–479, 2014. doi:10.1016/j.comgeo.2013.11.004.

[7] Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In *Proc. 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of *LNCS*, pages 553–564. Springer, 2014. doi:10.1007/978-3-319-13075-0_44.

[8] Tetsuo Asano, Wolfgang Mulzer, Günter Rote, and Yajun Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

[9] John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In Yuval Rabani, editor, *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 551–569. SIAM, 2012.

[10] Luis Barba, Matias Korman, Stefan Langerman, and Rodrigo I. Silveira. Computing a visibility polygon using few variables. *Comput. Geom. Theory Appl.*, 47(9):918–926, 2014. doi:10.1016/j.comgeo.2014.04.001.

[11] Luis Barba, Matias Korman, Stefan Langerman, Rodrigo I. Silveira, and Kunihiko Sadakane. Space-time trade-offs for stack-based algorithms. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 281–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.281.

[12] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sub-linear space, polynomial time algorithm for directed *s-t* connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

[13] Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991.

[14] Paul Beame, Michael E. Saks, Xiaodong Sun, and Erik Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *J. ACM*, 50(2):154–195, 2003. doi:10.1145/636865.636867.

[15] Aharon Ben-Tal and Arkadi Nemirovski. Selected topics in robust convex optimization. *Math. Program.*, 112(1):125–158, 2008. doi:10.1007/s10107-006-0092-2.

[16] Jon Louis Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers*, 28(9):643–647, 1979.

[17] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd edition, 2008.

[18] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Rev.*, 53(3):464–501, 2011. doi:10.1137/080734510.

[19] René van Bevern. *Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications*. Universitätsverlag der TU Berlin, 2014.

[20] René v. Bevern, Sepp Hartung, Frank Kammer, Rolf Niedermeier, and Mathias Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In *Proc. 6th International Symposium on Parameterized and Exact Computation (IPEC 2011)*, volume 7112 of *LNCS*, pages 194–206. Springer, 2011. doi:10.1007/978-3-642-28050-4_16.

[21] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Comput. methods appl. mech. eng.*, 196(33):3190–3218, 2007.

[22] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

[23] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996. doi:10.1137/S0097539793251219.

[24] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

[25] Hans L. Bodlaender and Ton Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 544–555. Springer, 1991. doi:10.1007/3-540-54233-7_162.

[26] Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982. doi:10.1137/0211022.

[27] Hervé Brönnimann, John Iacono, Jyrki Katajainen, Pat Morin, Jason Morrison, and Godfried Toussaint. Space-efficient planar convex hull algorithms. *Theor. Comput. Sci.*, 321(1):25–40, 2004.

[28] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distrib. Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.

[29] Timothy M. Chan. Closest-point problems simplified on the RAM. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 472–473. ACM/SIAM, 2002.

[30] Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6(2):Article 26, 2010. doi:10.1145/1721837.1721842.

[31] Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007. doi:10.1007/s00454-006-1275-6.

[32] Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Faster, space-efficient selection algorithms in read-only memory for integers. In *Proc. 24th International Symposium on Algorithms and Computation (ISAAC 2013)*, volume 8283 of *LNCS*, pages 405–412. Springer, 2013. doi:10.1007/978-3-642-45030-3_38.

[33] Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Selection and sorting in the "restore" model. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 995–1004. SIAM, 2014. doi:10.1137/1.9781611973402.74.

[34] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280 – 301, 2001. doi:http://dx.doi.org/10.1006/jagm.2001.1186.

[35] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

[36] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[37] D. Dai and C. Yu. A $(5 + \epsilon)$-approximation algorithm for minimum weighted dominating set in unit disk graph. *Theoret. Comput. Sci.*, 410(8-10):756–765, 2009.

[38] Jean Daligault and Stéphan Thomassé. On finding directed trees with many leaves. In *Proc. 4th International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *LNCS*, pages 86–97. Springer, 2009.

[39] Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *Proc. 22nd Annual European Symposium on Algorithms (ESA 2014)*, volume 8737 of *LNCS*, pages 284–295. Springer, 2014. doi:10.1007/978-3-662-44777-2_24.

[40] Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. *Theory Comput. Syst.*, 53(4):669–689, 2013. doi:10.1007/s00224-013-9469-9.

[41] Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.

[42] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. *J. Algorithms*, 48(1):2–15, 2003. doi:10.1016/S0196-6774(03)00043-9.

[43] A. Dhawan, C. T. Vu, A. Zelikovsky, Y. Li, and S. K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. In *Proc. 7th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and 2nd ACIS International Workshop on Self-Assembling Wireless Networks (SNPD-SAWN 2006)*, pages 285–289, 2006.

[44] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[45] Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. *Theory Comput. Syst.*, 56(1):220–250, 2015. doi:10.1007/s00224-014-9591-3.

[46] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[47] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[48] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 717–736. SIAM, 2013. doi:10.1137/1.9781611973105.52.

[49] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for *st*-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999. doi:10.1137/S0097539795295948.

[50] Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proc. 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 383–392. ACM, 2014. doi:10.1145/2591796.2591865.

[51] Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

[52] Amr Elmasry, Daniel Dahl Juhl, Jyrki Katajainen, and Srinivasa Rao Satti. Selection from read-only memory with limited workspace. *Theor. Comput. Sci.*, 554:64–73, 2014. doi:10.1016/j.tcs.2014.06.012.

[53] Amr Elmasry and Frank Kammer. Space-efficient plane-sweep algorithms. *Computing Research Repository (CoRR)*, abs/1507.01767, 2016.

[54] Thomas Erlebach, Tom Grant, and Frank Kammer. Maximising lifetime for fault-tolerant target coverage in sensor networks. *Sustain. Computing: Inform. Systems*, 1(3):213 – 225, 2011. doi:http://dx.doi.org/10.1016/j.suscom.2011.05.005.

[55] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proc. 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Part I*, volume 9134 of *LNCS*, pages 444–455. Springer, 2015. doi:10.1007/978-3-662-47672-7_36.

[56] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Unpublished journal version*, 2016.

[57] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theor. Comput. Sci.*, 613:51–64, 2016. doi:10.1016/j.tcs.2015.11.025.

[58] Thomas Erlebach and Matús Mihalák. A $(4+\epsilon)$-approximation for the minimum-weight dominating set problem in unit disk graphs. In *Proc. 7th International Workshop on Approximation and Online Algorithms (WAOA 2009)*, LNCS 5893, pages 135–146. Springer, 2009.

[59] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

[60] Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for cluster editing. In *Proc. 16th International Symposium on Fundamentals of Computation Theory (FCT 2007)*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007. doi:10.1007/978-3-540-74240-1_27.

[61] Mike Fellows. Table of FPT races. http://fpt.wikidot.com/fpt-races.

[62] Matteo Fischetti and Michele Monaci. Light robustness. In *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, volume 5868 of *LNCS*, pages 61–84. Springer, 2009. doi:10.1007/978-3-642-05465-5_3.

[63] Fedor V. Fomin, Sang-il Oum, and Dimitrios M. Thilikos. Rank-width and tree-width of H-minor-free graphs. *Eur. J. Comb.*, 31(7):1617–1628, 2010. doi:10.1016/j.ejc.2010.05.003.

[64] Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987.

[65] Vincent Froese, Iyad A. Kanj, André Nichterlein, and Rolf Niedermeier. Finding points in general position. *Computing Research Repository (CoRR)*, abs/1508.01097, 2015.

[66] Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *Eur. J. Oper. Res.*, 235(3):471–483, 2014. doi:10.1016/j.ejor.2013.09.036.

[67] M. R. Garey and D. S. Johnson. Computers and intractability, a guide to the theory of NP-completeness. W. H. Freeman and Co., 1979.

[68] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

[69] Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo I. Silveira. Removing local extrema from imprecise terrains. *Comput. Geom.*, 45(7):334–349, 2012. doi:10.1016/j.comgeo.2012.02.002.

[70] Christopher Green. Classics in the history of psychology. http://psychclassics.yorku.ca/Lovelace/lovelace.htm, 2013.

[71] Martin Grohe. Descriptive and parameterized complexity. In *Proc. 13th International Workshop on Computer Science Logic (CSL 1999)*, volume 1683 of *LNCS*, pages 14–31. Springer, 1999.

[72] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Trans. Algorithms*, 4(3), 2008. doi:10.1145/1367064.1367070.

[73] Qian-Ping Gu and Hisao Tamaki. Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in $O(n^{1+\epsilon})$ time. *Theor. Comput. Sci.*, 412(32):4100–4109, 2011. doi:10.1016/j.tcs.2010.07.017.

[74] Qian-Ping Gu and Gengchun Xu. Near-linear time constant-factor approximation algorithm for branch-decomposition of planar graphs. In *Proc. 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2014)*, volume 8747 of *LNCS*, pages 238–249. Springer, 2014. doi:10.1007/978-3-319-12340-0_20.

[75] Jiong Guo. A more effective linear kernelization for cluster editing. *Theoret. Comput. Sci.*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.

[76] Torben Hagerup. Simpler linear-time kernelization for planar dominating set. In *Proc. 6th International Symposium on Parameterized and Exact Computation (IPEC 2011)*, volume 7112 of *LNCS*, pages 181–193. Springer, 2011. doi:10.1007/978-3-642-28050-4_15.

[77] Torben Hagerup and Frank Kammer. Succinct choice dictionaries. *Computing Research Repository (CoRR)*, arXiv:1604.06058 [cs.DS], 2016.

[78] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory.* Languages and Compututions. Addison-Wesley, 1979.

[79] Y. Huang, X. Gao, Z. Zhang, and W. Wu. A better constant-factor approximation for weighted dominating set in unit disk graph. *J. Comb. Optim.*, 18(2):179–194, 2009.

[80] Frank Kammer. A linear-time kernelization for the rooted k-leaf outbranching problem. *Discret. Appl. Math.*, 193:126–138, 2015. doi:10.1016/j.dam.2015.04.028.

[81] Frank Kammer, Maarten Löffler, Paul Mutser, and Frank Staals. Practical approaches to partially guarding a polyhedral terrain. In *Proc. 8th International Conference on Geographic Information Science (GIScience 2014)*, volume 8728 of *LNCS*, pages 318–332. Springer, 2014. doi:10.1007/978-3-319-11593-1_21.

[82] Frank Kammer and Hanjo Täubig. Graph connectivity. www.informatik.uni-augsburg.de/thi/personen/kammer/Graph_Connectivity.pdf, 2004.

[83] Frank Kammer and Torsten Tholey. The k-disjoint paths problem on chordal graphs. In *Proc. 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *LNCS*, pages 190–201, 2009. doi:10.1007/978-3-642-11409-0_17.

[84] Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. In *Proc. 23th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 683–698. SIAM, 2012.

[85] Frank Kammer and Torsten Tholey. The complexity of minimum convex coloring. *Discrete Applied Mathematics*, 160(6):810–833, 2012. doi:10.1016/j.dam.2011.09.022.

[86] Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theor. Comput. Sci.*, 645:60–90, 2016. doi:10.1016/j.tcs.2016.06.040.

[87] Ton Kloks, Chuan-Min Lee, and Jiping Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In *Proc. 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002)*, volume 2573 of *LNCS*, pages 282–295. Springer, 2002. doi:10.1007/3-540-36379-3_25.

[88] Matsuo Konagaya and Tetsuo Asano. Reporting all segment intersections using an arbitrary sized work space. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 96-A(6):1066–1071, 2013.

[89] Matias Korman, Wolfgang Mulzer, Andre van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein. Time-space trade-offs for triangulations and Voronoi diagrams. In *Proc. 14th Algorithms and Data Structures Symposium (WADS 2015)*, volume 9214 of *LNCS*, pages 482–494. Springer, 2015.

[90] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proc. 42nd ACM Symposium on Theory of Computing, (STOC 2010)*, pages 513–522. ACM, 2010. doi:10.1145/1806689.1806760.

[91] Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 532–543. Springer, 1991. doi:10.1007/3-540-54233-7_161.

[92] Murray Langton. Wikipedia page on algorithmic efficiency. https://en.wikipedia.org/wiki/Algorithmic_efficiency, 2013.

[93] Jean B. Lasserre. Min-max and robust polynomial optimization. *J. Global Optimization*, 51(1):1–10, 2011.

[94] Maarten Löffler and Marc J. van Kreveld. Largest and smallest tours and convex hulls for imprecise points. In *Proc. 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059 of *LNCS*, pages 375–387. Springer, 2006. doi:10.1007/11785293_35.

[95] J. Łukasiewicz and A. Tarski. Untersuchungen über den aussagenkalkül. *Comp. Rend. Soc. Sci. et Lettres Varsovie Cl. III 23*, pages 30–50, 1930.

[96] Mirela Marta, Yinying Yang, and Mihaela Cardei. Energy-efficient composite event detection in wireless sensor networks. In *Proc. 4th International Conference on Wireless Algorithms, Systems, and Applications (WASA 2009)*, volume 5682 of *LNCS*, pages 94–103. Springer, 2009. doi:10.1007/978-3-642-03417-6_10.

[97] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Computing Research Repository (CoRR)*, abs/1503.00278:1–42, 2015.

[98] Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. In *Proc. 39th International Symposium on Mathematical Foundations of Computer Science 2014 (MFCS 2014), Part II*, volume 8635 of *LNCS*, pages 553–564, 2014. doi:10.1007/978-3-662-44465-8_47.

[99] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12(3):315–323, 1980.

[100] J. Ian Munro and Venkatesh Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996.

[101] G. L. Nemhauser and L. E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi:10.1007/BF01580444.

[102] Nils J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28(1):71–87, 1986. doi:10.1016/0004-3702(86)90031-7.

[103] Sang-il Oum. Rank-width is less than or equal to branch-width. *J. Graph Theory*, 57(3):239–244, 2008. doi:10.1002/jgt.20280.

[104] Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743455.

[105] Fàbio Protti, Maise Dantas da Silva, and Jayme Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory Comput. Syst.*, 44:91–104, 2009.

[106] Venkatesh Raman and Sarnath Ramnath. Improved upper bounds for time-space trade-offs for selection. *Nord. J. Comput.*, 6(2):162–180, 1999.

[107] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pages 221–228. ACM, 1992. doi:10.1145/129712.129734.

[108] John Richardson, Claudia Gorbman, and Carol Vernallis. The oxford handbook of new audiovisual aesthetics. *Oxford University Press*, 2015.

[109] Neil Robertson and Paul D. Seymour. Graph minors III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

[110] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

[111] Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

[112] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

[113] H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik in Saarbrücken, 1998.

[114] Peter Sanders and Dennis Schieferdecker. Lifetime maximization of monitoring sensor networks. In *Proc 6th International Workshop on Algorithms for Sensor Systems, Wireless Ad Hoc Networks, and Autonomous Mobile Entities (ALGOSENSORS 2010)*, volume 6451 of *LNCS*, pages 134–147. Springer, 2010.

[115] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.*, 4:177–192, 1970.

[116] Christian Scheideler. Models and techniques for communication in dynamic networks. In *Proc. 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 2285 of *LNCS*, pages 27–49. Springer, 2002. doi:10.1007/3-540-45841-7_2.

[117] Stefan Schirra. Robustnes and precision issues in geometric computation. *Handbook of Computational Geometry*, pages 597–632, 1999.

[118] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

[119] R.E. Stearns, J. Hartmanis, and P.M. Lewis. Hierarchies of memory limited computations. In *Proc. 6th IEEE Conf. of Switching Circuit Theory and Logical Design*, pages 179–190, 1965.

[120] My T. Thai, Feng Wang, David Hongwei Du, and Xiaohua Jia. Coverage problems in wireless sensor networks: designs and analysis. *Intern. J. Sens. Netw.*, 3(3):191–200, 2008.

[121] C.T. Vu, R.A. Beyah, and Y. Li. Composite event detection in wireless sensor networks. In *Proc. 26th IEEE International Performance, Computing, and Communications Conference (IPCCC 2007)*, pages 264–271, 2007.

[122] World Data Center Climate. German Climate Computing Centre (DKRZ GmbH), Hamburg. http://wdc-climate.de.

[123] Andrew Chi-Chih Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM J. Comput.*, 23(5):966–975, 1994.

[124] Lotfi A. Zadeh. Fuzzy sets. *Inform. Control*, 8(3):338–353, 1965. doi:10.1016/S0019-9958(65)90241-X.

[125] Qun Zhao and Mohan Gurusamy. Lifetime maximization for connected target coverage in wireless sensor networks. *IEEE/ACM Trans. Netw.*, 16:1378–1391, 2008.

[126] F. Zou, Y. Wang, X.H. Xu, X. Li, H. Du, P. Wan, and W. Wu. New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs. *Theoret. Comput. Sci.*, 412:198–208, 2011.