Weak Kernels

Haitao Jiang* Chihao Zhang† Binhai Zhu‡

November 18, 2010

Abstract

In this paper, we formalize a folklore concept and formally define *weak kernels* for (NP-hard) search problems, which is about search space reduction and stands as a new generic technique for designing FPT algorithms. We show that weak kernels are different from the (traditional) kernels for decision problems, by exhibiting an example out of P such that its decision version has no kernel while the equivalent search problem has a weak kernel. We show a few applications of weak kernels, for which a traditional kernelization seems hard to apply. Among them, we present the first FPT algorithm for the famous Sorting by Minimum Unsigned Reversals problem.

*Department of Computer Science, Montana State University, Bozeman, MT 59717, USA. Email: htjiang@cs.montana.edu.

†Department of Computer Science, Shanghai Jiao Tong University, Shanghai 200030, China. Email: chihao.zhang@gmail.com.

‡Corresponding author. Department of Computer Science, Montana State University, Bozeman, MT 59717, USA. Email: bhz@cs.montana.edu.

1 Introduction

In the last four decades, we have seen the huge advance of NP-completeness [12, 27, 19]. Nowadays, NP-complete problems appear in almost all the areas which involves combinatorial optimization, for example in computational biology and bioinformatics. As from the beginning a lot of people tended to believe P$\neq$NP (at least it seems to be hard to prove or disprove it), people immediately started to investigate different ways to handle NP-hard problems. Up to today, the two most popular ways to handle NP-hard problems, among researchers in algorithm design, are approximation algorithms and exact (or FPT) algorithms, which were started with the seminal works of Johnson [25] and Tarjan and Trojanowski [31] respectively. (Using heuristic methods to hand NP-hard problems, like evolutionary computation, is beyond this paper.)

In some areas like computational biology and bioinformatics, the data usually contain errors. On top of this, if we design a factor-2 approximation to handle these data, whatever result we got is not appealing to biologists. So, to make approximation algorithms useful for these applications, the approximation factors must be very close to one. Then, naturally, FPT algorithm pops up as a natural alternative for handling these problems. The two applications we will discuss in this paper all originate from computational biology.

On the other hand, the theory of fixed-parameter computation has been developed rigorously in the last two decades. The first textbook was published in 1999 by Downey and Fellows [14] and another couple were published in the last several years (e.g., [17]). Interested readers are referred to [20] for further details and references.

In designing FPT algorithms, kernelization is one of the most fundamental techniques for decision problems. Loosely speaking, kernelization is really *data reduction*; i.e., with kernelization one reduces the problem instance size (kernel size) to a level so small that one could even apply a brute-force method. Sometimes, even if the kernel size is slightly bigger (say 2^k) so that a brute-force method is inappropriate, one can still make use of it with integer linear programming or branch-and-bound to obtain almost optimal solutions in a reasonable amount of time [20].

In this paper, we formalize a folklore method and formally define *weak kernels* and weak kernelization for *search* problems. Again, loosely speaking, when viewing an NP-hard optimization problem as a search problem (like for Vertex Cover, we are really searching for a set of k vertices, among n input vertices, so that deleting the k vertices leaves the resulting graph edge-less), weak kernelization is really about *search space reduction*. We show that in general weak kernels and kernels are not equivalent. This is done by showing an example out of P such that its search version has a weak kernel but its equivalent decision version has no kernel. (We comment that this "search vs decision" question has been considered in the complexity theory before, as early as in 1974 by

Valiant [33]. Interested readers are referred to [5] for the further development.)

The purpose for defining the weak kernels concept, on the other hand, is more on helping us design FPT algorithms more easily. In other words, weak kernelization should be considered as a new generic method for designing FPT algorithms efficiently. Here, we show an application of weak kernels to two problems, all known to be NP-complete, for which we compute the corresponding weak kernels efficiently (hence design efficient FPT algorithms). Among the two problems, Sorting with Minimum Unsigned Reversals is a famous problem in computational biology and we do not know of any non-trivial kernelization or FPT algorithm for it. We show that Sorting with Minimum Unsigned Reversals has a weak kernel of size $4k$, hence an FPT algorithm running in $O(2^{4k}n + n \log n)$ time (and with a more detailed analysis, in $O(2^{2k}n + n \log n)$ time).

2 Kernels vs Weak Kernels

2.1 Preliminaries

Basically, a fixed-parameter tractable (FPT) algorithm for a *decision* problem Π with solution value k is an algorithm which solves the problem in $O(f(k)n^c)$ time, where f is any function only on k, n is the input size and c is some fixed constant not related to k. FPT also stands for the set of problems which admit such an algorithm [14]. (Let Σ be a finite alphabet. In the languages of [17], a *parameterized problem* (Q, κ) is composed of a set $Q \subseteq \Sigma^*$ of strings and a parameterization κ of Σ^* (which maps Σ^* to $\mathbb{N}$). An FPT algorithm for (Q, κ) is then an algorithm which solves it in $f(\kappa(x)) \cdot p(|x|)$ time, where x is the input length, f is any computation function and p is a polynomial function.)

Kernelization is a polynomial time transformation that transforms a problem instance (I, k) to another instance (I', k') such that (1) (I, k) is a yes-instance iff (I', k') is also a yes-instance; (2) $k' \leq k$; and (3) $|I'| \leq f(k)$ for some function $f(-)$. (I', k') is typically called a *kernel* of the problem, with size $|I'|$. It is easy to see that if a problem has a kernel then it is in FPT; moreover, every problem in FPT has a kernel. From this, kernelization is a way to perform data-reduction with performance guarantee, "a humble strategy for coping with hard problems, almost universally employed" [15]. More fundamental details on FPT algorithms can be found in [14, 17].

Recently, Bodlaender *et al.* conducted a seminal work by showing that a class of important FPT problems cannot have polynomial (e.g., $O(k^2)$ size) kernels unless the polynomial hierarchy (PH) collapses to the third level (i.e., $\mathrm{PH} = \Sigma_p^3$) [6]. (The fundamental technique of this work, however, is adapted from [18].) One such problem is called k-LEAF OUT-BRANCHING (i.e., finding a rooted oriented spanning tree with at least k leaves in an input digraph $\mathcal{D}$) [16].

2.2 Weak Kernels

As illustrated in the introduction, we view weak kernelization as a way to reduce search space. In the following, we formalize the folklore concept and call it weak kernel. We also prove some of its basic properties.

Definition 1 (Search Problem) *Let Σ be the alphabet and $L \subseteq \Sigma^*$ be a decidable language. A search problem w.r.t. L is a binary relation $R_L \subseteq \Sigma^* \times \Sigma^*$. $x \in L$ iff $\exists y \in \Sigma^*$ such that $R(x, y)$. We say a Turing machine T computes R if:*

- *If $x \in L$, then T accepts x with output $y \in \Sigma^*$ such that $R(x, y)$.*
- *If $x \notin L$, then T rejects x.*

Intuitively, two strings $x, y \in \Sigma^*$ such that $R(x, y)$ means that y is a witness of $x \in L$.
With this definition, a search problem is in NP if:

- There is a polynomial $p : \mathbb{N} \to \mathbb{N}$, for any $x, y \in \Sigma^*$, $R(x, y)$ implies $|y| \leq p(|x|)$.
- For any $x, y \in \Sigma^*$, $R(x, y)$ can be decided in PTIME.

Then the search space of a search problem is a language from which solution could be extracted.

Definition 2 (Search Space) *Given a search problem R_L w.r.t. a language L and $x \in \Sigma^*$, the search space of R_L is a language L' with two algorithms S and A such that $x \in L$ iff $S(x) \in L'$ and $R_L(x, A(S(x)))$.*

Definition 3 (Weak Kernel) *Let R_Q be a parameterized search problem over alphabet Σ with the underlying decision problem (Q, κ). Let T be a Turing machine that decides R and its runtime is bounded by a function f.*

A polynomial time computable function $W : \Sigma^* \to \Sigma^*$ is a weak kernelization of (Q, κ) if there exists an algorithm A_W such that $L_W := \{W(x) : x \in L\}$ is a search space of R_Q, moreover, for each $x \in \Sigma^*$, $W(x) = (w_1, w_2)$ with $|w_1| \leq h(\kappa(x))$, $|w_2| \leq q(f(|x| + \kappa(x)))$ and the runtime of $A_W(w_1, w_2)$ is bounded by $g(|w_1|) \times p(|w_2|)$ where g, h are arbitrary computable functions and p, q are polynomial functions. L_W is called the weak kernel.

Let W be a weak kernelization and $W(x) = (w_1, w_2)$ for some $x \in \Sigma^*$. We define the size of weak kernel w.r.t. x as $|w_1|$.

In the definition of weak kernelization, the runtime of A_W depends on two parts, say $g(|w_1|)$ and $p(|w_2|)$ where g is an arbitrary computable function and p is a polynomial function. In many practical cases, w_1 contains the essential information to obtain the solution and w_2 only deals with encoding.

For instance, for the parameterized Vertex Cover (p-Vertex-Cover) with instance $(G = (V, E), k)$, $|w_2|$=$O(k \log |V|) = O(k \log n)$. This issue was raised by Harnik and Naor before [21]. However, for our applications, as all the problems are in NP, this actual encoding blow-up can be almost always ignored. This is similar to the RAM model, in which one can store a vertex/integer using $O(1)$ space; but in theory we need to store $\log n$ bits for a vertex if there are n vertices to store.

However, in the following example, the w_2 part is used to verify the solution.

Example 1 P-SAT *has no polynomial kernelization unless* PH *collapses to its third level (i.e.,* $\mathrm{PH} = \Sigma_p^3$*) [18, 6]. But it has a weak kernelization such that* $W(x) = (\kappa(x), x)$.

2.3 Kernels $\neq$ Weak Kernels

Weak kernelization is somehow a generalization of kernelization to search problems. In essence, weak kernelization deals with problems that search for a witness. However, since in a decision problem, the solution is "Yes" or "No" and always different from its witness, these two notations are different if we directly change a search problem to decision one. We show below that for some logically equivalent decision and search problem, kernel and weak kernel cannot co-exist.

Example 2 *Let* $Q \notin \mathrm{P}$ *be some language over* Σ *such that for any* $x \in \Sigma^*$*, whether* $x \in Q$ *can be decided in* $f(|x|)$ *time. Define a search problem* R_Q *as below:*

- $\forall x \in \Sigma^*$*, if* $x \in Q$*, then* $(x, 1) \in R_Q$.
- $\forall x \in \Sigma^*$*, if* $x \notin Q$*, then* $(x, 0) \in R_Q$.

Let $\kappa(x) = 1$ *for every* $x \in \Sigma^*$*. Then* R_Q *has a weak kernelization but* (Q, κ) *has no kernelization.*

Proof. It is easy to see that (Q, κ) has no kernelization for otherwise an FPT algorithm for (Q, κ) would imply $Q \in \mathrm{P}$.

R_Q has a trivial weak kernelization that $W(x) = x1^{f(|x|)}$ for all $x \in \Sigma^*$. That is, $W(x)$ is x followed by $f(|x|)$ 1's. (Note that $w_1 = \emptyset$ in this case.) The algorithm A_W just tests $R(x, 0)$ and $R(x, 1)$. □

We comment that the above result is related to the "search vs decision" question in the traditional complexity theory; namely, under a complexity assumption, there is an associated search problem ρ in NP which cannot be reduced to its corresponding decision problem [5].

However, if the underlying decision problem for a search problem is in NP, then weak kernelization implies kernelization.

Proposition 1 *Let R_Q be a search problem with underlying parameterized decision problem (Q, κ), and $Q \in \mathrm{NP}$, then a weak kernelization for R_Q implies a kernelization for (Q, κ).*

Proof. Let W be a weak kernelization and $W(x) = (w_1, w_2)$ for $x \in \Sigma^*$. By the definition of weak kernel, $|w_1| \leq h(\kappa(x))$ and $|w_2| \leq q(f(|x| + \kappa(x)))$ for some computable function h and polynomial q; and furthermore, f is also a polynomial function since $Q \in \mathrm{NP}$. Then the search algorithm A_W whose runtime is $g(|w_1|) \times p(|w_2|)$ for some computable function g and polynomial p implies an FPT algorithm to decide Q. □

While the above proofs are not really difficult, they have interesting theoretical implications. For instance, for a problem unlikely to have a kernel (say k-Dominating Set, which is W[2]-complete), as long as it belongs to NP, it is equally unlikely to have a weak kernel. Therefore, for problems in NP, the true merit of the above concepts seems to be helping us design efficient FPT algorithms via weak kernels directly.

Through a private communication with Mike Fellows, the earliest idea of using weak kernels seems to be in [1], where Bonsma, Brüggemann and Woeginger showed that the MAX LEAF problem has a weak kernel of size $3.5k$. (Note that MAX LEAF is the complement of the Minimum Connected Dominating Set problem.) In the next section, we show two new applications of weak kernels.

3 Applications

We show below two examples of the applications of weak kernels. For both of them, we do not know of better kernel bounds. For the famous Sorting with Minimum Unsigned Reversals, this is the first non-trivial FPT algorithm.

The two minimization problems we consider are all known to be NP-complete: Minimum Co-Path Set and Sorting with Minimum Unsigned Reversals (SMUR). We will mainly focus on solving these problems with weak kernels. For some of these problems (e.g., Minimum Co-Path Set), it is possible to solve it with bounded search tree, on top of weak kernels. Yet in general it is still unknown whether bounded search tree is always more powerful than weak kernels.

3.1 Minimum Co-Path Set

In this subsection, we study the following problem called Minimum Co-Path Set. Given a simple undirected graph G, a *co-path set* is a set S of edges in G whose removal leaves a graph in which

every connected component is a path. In the Minimum Co-Path Set Problem, we need to compute a minimum co-path set in G.

The Minimum Co-Path Set Problem originates from radiation hybrid (Rh) mapping, which is a powerful technique for mapping unique DNA sequences onto chromosomes and whole genomes [9, 13, 28, 29]. In Rh mapping, chromosomes are randomly broken into small DNA fragments through gamma radiation. A (random) subset of these DNA fragments retain with healthy hamster cells and grow up to build up a hybrid cell line. This process is repeated many times and the co-retention rate of a pair of markers (labeled chromosomal loci) indicates their physical distance on the chromosome. In principle, when two markers x and y are close, the probability that x and y are broken by the gamma radiation is small, hence with a high probability they are either co-present in or co-absent from a DNA fragment.

A subset of markers that are co-present from DNA fragments is called a *cluster*. Let $V = \{1, 2, \cdots, n\}$ be a set of markers and let $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$ be a collection of clusters. The Radiation Hybrid Map Construction Problem is to compute a linear ordering of the markers in which the markers in each cluster C_i appear consecutively. In reality, a cluster might be formed with errors, so no such linear ordering might exist. In this case, one needs to remove the minimum number of clusters so that the leftover clusters admit a linear ordering. When $|C_i| = 2$ for all i, this is exactly the Minimum Co-Path Set Problem. Given a simple undirected graph $G = (V, E)$, each vertex in V corresponds to a marker, an edge $(u, v) \in E$ corresponds to a cluster $\{u, v\}$.

In [9], the Minimum Co-Path Set Problem was shown to be NP-complete [19]. The proof is by a reduction from the Hamiltonian Path problem, with each edge (u, v) being converted to a cluster $\{u, v\}$. It is easy to see that there is a Hamiltonian Path in the input graph G if and only if one has to delete exactly $|E| - n + 1$ clusters. A factor-2 approximation was also proposed in [9], which was recently improved to 10/7 [11]. (The counterpart of the Minimum Co-Path Set Problem is the well-known *Minimum Path Cover* problem [34] and will not be covered here.) Let k be the minimum number of edges deleted for the problem. We show in this subsection that the Minimum Co-Path Set Problem is in FPT; in fact, it has a linear weak kernel of size at most $5k$, hence the problem can be solved efficiently in $O(2^{3.61k}(n + k))$ time. In the following, we present the technical details.

If some connected component of G has maximum vertex degree at most 2 then the problem is trivially solvable for that component. So from now on we assume that each connected component of G has maximum vertex degree at least 3. Moreover, in the solution a single vertex could also be considered as a (degenerate) path. The following lemma is easy to prove.

Lemma 1 *There is a solution R for the minimum co-path set such that R contains only edges incident to some vertices of degree at least 3 in G.*

Proof. Assume to the contrary that a solution R contains some edge (x, y) such that both x and y have degree at most two in G. Let $G - R$ be the graph obtained from G by deleting all the edges in R. When both x and y have degrees at most 2, if (x, y) is in R then putting it back to $G - R$ would have two possibilities: (1) make each connected component of $(G - R) \cup \{(x, y)\}$ a path, or (2) create some cycle in $(G - R) \cup \{(x, y)\}$. In case (1), it contradicts the optimality of R. In case (2), (x, y) is on some cycle in G. Hence we can find an edge (x', y') on this cycle which is incident to some vertex of degree at least 3 in G. Then we simply swap (x, y) with (x', y') in R. It is easy to see that repeating this process we can eventually have a new solution R' such that $|R'| = |R|$ and R' contains only edges incident to some vertices of degree at least 3 in G. □

Now let D be a solution for the minimum co-path set such that D contains only edges incident to some vertices of degree at least 3 in G. The above lemma implies a simple weak kernelization procedure.

1. Identify the vertices of G with degree at least 3. Let this set be $V_3(G)$.
2. Let the set of edges which are incident to some vertices in $V_3(G)$ be $E_3(G)$.
   Return $E_3(G)$ as a weak kernel.

We have the following lemma.

Lemma 2 *The Minimum Co-Path Set Problem has a solution of size k if and only if the solution can be obtained by deleting k edges in $E_3(G)$.*

Proof. We only need to show the 'only-if' part as the other part is obvious. By Lemma 1, we do not need to include any edge in D which is incident to vertices of degree only one or two. □

It remains to show the weak kernel size (i.e., the size of $E_3(G)$). We have the following lemma.

Lemma 3 *Let $k = |D|$, then $|E_3(G)| \leq 5k$. In other words, the size of the weak kernel of the Minimum Co-Path Set Problem is $5k$.*

Proof. From Lemma 2, we know that the k edges of D can be found in $E_3(G)$. After these k edges in D are deleted from G, $G - D$ is only composed of paths, i.e., the degrees of vertices in $G - D$ are at most 2. In other words, the edges in $E_3(G) - D$ must also be incident to vertices in $G - D$ of degree at most 2 (note that these vertices originally are all in $V_3(G)$). As the k edges in D are incident to at most $2k$ vertices in $V_3(G)$, $|V_3(G)| \leq 2k$. Therefore, we have at most $4k$ edges in $E_3(G) - D$. Counting the k edges in D back, we have $|E_3(G)| = |E_3(G) - D| + |D| \leq 4k + k = 5k$. □

With the above lemmas, it is easy to have an FPT algorithm for the Minimum Co-Path Set Problem. First, if $|V_3(G)| > 2k$ or $|E_3(G)| > 5k$ then we can simply return NO. Otherwise, among

the (at most) $5k$ edges in $E_3(G)$, select all combinations of k edges to delete. For each set of k edges selected, delete them from G and check whether the resulting graph is composed of paths only (using standard linear time graph algorithms like depth-first search). If we fail to find such a set, then return 'No solution of size k'; otherwise, just return the computed set of edges as D. The time complexity of the algorithm is dominated by checking $\begin{pmatrix} 5k \\ k \end{pmatrix} \approx 2^{3.61k}$ solutions. We have the following theorem.

Theorem 1 *Let k be the size of the minimum co-path set. The Minimum Co-Path Set Problem has a weak kernel of size $5k$, hence can be solved in $O(2^{3.61k}(n+k))$ time.*

3.2 Sorting with Minimum Unsigned Reversals

Sorting with Minimum Unsigned Reversals (SMUR) is a famous problem in computational biology, more specifically, in computational genomics. Given a genome H composed of a sequence of n distinct genes (also formulated as a permutations of n integers $\{1, 2, \cdots, n\}$), i.e., assume that $H = s_1 s_2 \cdots s_i s_{i+1} \cdots s_{j-1} s_j \cdots s_n$, a *reversal* operation on the segment $s_i s_{i+1} \cdots s_{j-1} s_j$ transforms H into $H' = s_1 s_2 \cdots s_j s_{j-1} \cdots s_{i+1} s_i \cdots s_n$. The problem Sorting with Minimum Unsigned Reversals is to use the minimum number of reversals to convert H into the identity permutation $I = 123 \cdots n$. Example: Given $H = 15342$, we can use two signed reversals to first change it to 15432 and finally to 12345.

When the genes are signed, we have a similar problem Sorting with Minimum Signed Reversals. Given a signed genome H^- composed of a sequence of n distinct (signed) genes (also formulated as a signed permutations of n integers $\{1, 2, \cdots, n\}$), i.e., $H^- = t_1 t_2 \cdots t_i t_{i+1} \cdots t_{j-1} t_j \cdots t_n$, a *signed reversal* operation on the segment $t_i t_{i+1} \cdots t_{j-1} t_j$ transforms H^- into $H'' = t_1 t_2 \cdots - t_j - t_{j-1} \cdots - t_{i+1} - t_i \cdots t_n$. The problem Sorting with Minimum Signed Reversals is to use the minimum number of signed reversals to convert H^- into the identity permutation $I = 123 \cdots n$. Example: Given $H = 1-534-2$, we can use two signed reversals to first change it to $1-5-4-3-2$ and finally to 12345. (Note that in the literature it is also acceptable to convert H^- to $-I = -n \cdots -3 -2 -1$. We can enforce that H^- is converted to I by adding two auxiliary genes, i.e., $0H^-(n+1)$. This is a known trick in computational genomics.)

SMUR was shown to be NP-complete by Caprara [7] and the best approximation algorithm has a factor 1.375 [2]. However, no non-trivial FPT algorithm is known for the problem. The trivial solution is to use a bounded search tree algorithm which runs in roughly $O(k^{O(k)}n)$ time. We show below that with weak kernels, a much faster FPT algorithm can be designed.

We use Sorting with Minimum Signed Reversals as a subroutine for SMUR. Unlike SMUR,

Sorting with Minimum Signed Reversals can be solved in polynomial time [23, 26, 32], with the best running time being $O(n \log n)$ [30]. Computing the minimum signed reversal distance, however, can be done in linear time [4]. Let H be the (unsigned) genome to be sorted. It is easy to see that each reversal can eliminate at most two breakpoints. (In this case a breakpoint is a 2-substring $\langle i, j\rangle$ of H such that $|j - i| \neq 1$.) Hence, if the optimal solution size is k, there would be at most $2k$ breakpoints in H. In other words, there are at most $4k$ genes which are in some breakpoints. Let H_k be the set of such (at most) $4k$ genes. H_k is the weak kernel in this application.

Given H, let a maximal substring B of H composed of at least two consecutive adjacencies be called a *block*, with the first and last letters called *head* and *tail* of the block respectively. (We also say that the head and the tail are *adjacent through the block* B in H.) Example: $H = \langle 0, 5, 7, 8, 10, 1, 2, 3, 4, 9, 6, 11\rangle$, $B = \langle 1, 2, 3, 4\rangle$ is a block with head 1 and tail 4. 7 and 8 are in H_7 but form an adjacency in H. 1 and 4 are adjacent through the block B in H. Following [22], there is an optimal SMUR solution for H which does not cut any block.

Let H_k^- be the set of signed genomes obtained by adding $+/-$ signs on these genes (involved in some breakpoints) in H_k. (Following [22], if two such genes in H_k are the head and tail of a block B, then all the genes in B should be given the same sign, i.e., either all positive or all negative.) It is easily seen that $|H_k^-| \leq 2^{4k}$. Moreover, we have the following lemma.

Lemma 4 *There is a solution of k unsigned reversals for sorting H if and only if the solution can be found by sorting some sequence in H_k^- with k signed reversals.*

Proof. If there is a solution of k unsigned reversals for sorting H, then we can trace these k reversals backwards and each time add signs accordingly. For example, assume that the last reversal to obtain $\langle 0, 1, 2, 3, 4, 5\rangle$ is $\langle \underline{3, 2}\rangle$, then for sorting by signed reversals the second last signed genome is $\langle 0, 1, -3, -2, 4, 5\rangle$. It is easily seen that after repeating this process k times, we have a signed genome H'' in H_k^-. Certainly, one can sort H'' by k signed reversals.

On the other hand, if there are k signed reversals which sorts some genome in H_k^-, say H'', one can ignore the negative signs in H'' (to obtain H) and perform the same k (unsigned) reversals to sort H into I. □

Theorem 2 *Sorting with Minimum Unsigned Reversals has a weak kernel of size $4k$, hence can be solved in $O(2^{2k}n + n \log n)$ time.*

Proof. We first show a bound of $O(2^{4k}n + n \log n)$, which is straightforward from the $4k$ weak kernel. First, following Lemma 4, the weak kernelization is easy: identify all the blocks in H and return (H, H_k, k). For each possible signed genome in H_k^- (obtained from H_k by adding some

negative signs), we use the $O(n)$ time algorithm in [4] to check whether it can be sorted with k signed reversals. If so, we can compute accordingly the k signed reversals using the algorithm by Swenson *et al.* [30], to obtain the k (unsigned) reversals to sort H in $O(n \log n)$ time. If no valid solution is found, we report NO. This algorithm clearly runs in $O(2^{4k}n + n \log n)$ time.

By a more detailed analysis (i.e., we do not have to try all possible ways to sign genes in H_k), the running time of the above algorithm can be improved to $O(2^{2k}n + n \log n)$ time. Now let the genes in H_k form a total of z adjacencies (possibly through some blocks). Following [22], if two such genes form an adjacency in H, obviously they have to be given the same signs, i.e., either both positive or both negative. If two such genes form an adjacency through some block B in H, all the genes in B need to have the same signs. So the total number of ways to sign genes in H_k is bounded by

$$2^z \times 2^{(4k-2z)/2-1} = 2^{2k-1}.$$

Hence we have an FPT algorithm with running time $O(2^{2k}n + n \log n)$. □

We comment that, for the related Sorting with Minimum Unsigned Translocation problem, exactly the same idea can be applied to obtain a weak kernel of size $4k$, hence an FPT algorithm with running time $O(2^{2k}n + n^2)$. The relevant details can be found in [35, 3] (or from the references therein).

4 Concluding Remarks

We formally introduce a new (somehow a previous folklore) concept called weak kernels for fixed-parameter computation and prove some interesting properties of weak kernels. We also show some interesting applications with weak kernels. We believe that for certain problems weak kernels are more flexible and possibly more powerful than the traditional kernels. This is certainly the case with our two applications, especially the famous Sorting with Minimum Unsigned Reversals (SMUR) problem. We know of no FPT algorithm which runs close to $O^*(2^{4k})$ time for SMUR. It turns out that weak kernels can also be applied to obtain an efficient FPT algorithm for the problem of sorting linear genomes under the unsigned DCJ distance [24]. It would be interesting to see more applications of weak kernels.

Acknowledgments

This research is partially supported by NSF of China under project 60928006. We also thank Yijia Chen, Mike Fellows and Angsheng Li for several valuable comments.

References

[1] P. Bonsma, T. Brüggemann and G. Woeginger. A fast FPT algorithm for finding spanning trees with many leaves. In *Proc. 28th Intl. Symp. on Mathematical Foundations of Computer Science (MFCS'03), pages 259-268, 2003.*

[2] P. Berman, S. Hannenhalli and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proc. 10th European Symp. on Algorithms (ESA'02), pages 200-210, Rome, Italy, Sep, 2002*.

[3] A. Bergeron, J. Mixtacki and J. Stoye. On sorting by translocation. In *Proc. 9th Intl. Conf. on Research in Comput. Molecular Biology (RECOMB'05)*, pages 615-629, 2005.

[4] D. Bader, B. Moret and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J. of Computational Biology*, 8:483–491, 2001.

[5] M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM J. Comput.*, 23:97–119, 1994.

[6] H. Bodlaender, R. Downey, M. Fellows and D. Hermelin. On problems without polynomial kernels. In *Proc. 35th Intl. Colloquium on Automata, Languages and Programming (ICALP'08), pages 563-574, 2008.*

[7] A. Caprara. Sorting by reversals is difficult. In *Proc. 1st Intl. Conf. on Research in Comput. Molecular Biology (RECOMB'97)*, pages 75-83, 1997.

[8] Z. Chen, B. Fu, M. Jiang, and B. Zhu. On recovering syntenic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18:307–318, 2009.

[9] Y. Cheng, Z. Cai, R. Goebel, G. Lin and B. Zhu. The radiation hybrid map construction problem: recognition, hardness, and approximation algorithms. *Unpublished Manuscript*, 2008.

[10] Y. Chen and J. Flum. A logic for PTIME and a parameterized halting problem. In *Proc. 24th Annl. IEEE Symp. on Logic in Computer Science (LICS'09)*, pages 397-406, 2009.

[11] Z. Chen, G. Lin and L. Wang. An approximation algorithm for the minimum co-path set problem. *Algorithmica*, to appear, 2010.

[12] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symp. on Theory of Computing (STOC'71)*, pages 151-158, 1971.

[13] D.R. Cox, M. Burmeister, E.R. Price, S. Kim, and R.M. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high resolution maps of mammalian chromosomes. *Science*, 250:245–250, 1990.

[14] R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag, 1999.

[15] M. Fellows. The lost continent of polynomial time: preprocessing and kernelization. In *Proc. 2nd Intl. Workshop on Parameterized and Exact Computation (IWPEC'06)*, LNCS 4169, pages 276-277, 2006.

[16] H. Fernau, F. Fomin, D. Lokshtanov, D. Raible, S. Saurabh and Y. Villanger. Kernel(s) for problems with no kernel: on out-trees with many leaves. In *Proc. 26th Intl. Symp. on Theoretical Aspects of Computer Science (STACS'09)*, pages 421-432, 2009.

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer-Verlag. 2006.

[18] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proc. 40th ACM Symp. Theory of Computation (STOC'08), pages 133-142, Victoria, Canada, 2008.*

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[20] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38:31-45. 2007.

[21] D. Harnik and M. Naor. On the compressibility of NP instances and cryptographic applications. In *Proc. 47th IEEE Symp. Foundations of Computer Science (FOCS'06), pages 719-728, Berkeley, CA, 2006.*

[22] S. Hannenhalli and P. Pevzner. To cut...or not to cut (Applications of comparative physical maps in molecular evolution). In *Proceedings of the 7th ACM-SIAM Symp. on Discrete Algorithms (SODA'96)*, pages 304-313, 1996.

[23] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, **46**(1):1–27. 1999.

[24] H. Jiang, B. Zhu and D. Zhu. Algorithms for sorting linear genomes under the unsigned DCJ distance. *Bioinformatics*, submitted for publication, 2010.

[25] D. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Sys. Sciences*, 9:256–278, 1974.

[26] H. Kaplan, R. Shamir and R. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Comput.*, 29:880–892, 1999.

[27] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, NY, pages 85–103, 1972.

[28] C.W. Richard, D.A. Withers, T.C. Meeker, S. Maurer, G.A., Evans, R.M. Myers, and D.R. Cox. A radiation hybrid map of the proximal long arm of human chromosome 11, containing the multiple endocrine neoplasia type 1 (MEN-1) and bcl-1 disease loci. *American J. of Human Genetics*, 49:1189–1196, 1991.

[29] D. Slonim, L. Kruglyak, L. Stein, and E. Lander. Building human genome maps with radiation hybrids. *J. of Computational Biology*, 4:487–504, 1997.

[30] K. Swenson, V. Rajan, Y. Lin, and B. Moret. Sorting signed permutations by inversions in $O(n \log n)$ time. In *Proc. RECOMB'09*, LNCS 5541, pages 386-399, 2009.

[31] R. Tarjan and A. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6:537–546, 1977.

[32] E. Tannier and M-F. Sagot. Sorting by reversals in subquadratic time. In *Proc. 15th Symp. Combinatorial Pattern Matching (CPM'04)*, Istanbul, Turkey, pages 1-13, July, 2004.

[33] L. Valiant. On the relative complexity of checking and evaluating. *University of Leeds Technical Report LS29JT, October*, 1974.

[34] S. Vishwanathan. An approximation algorithm for the asymmetric travelling salesman problem with distance one and two. *Information Processing Letters*, 44:297–302, 1992.

[35] L. Wang, D. Zhu, X. Liu and S. Ma. An $O(n^2)$ algorithm for signed translocation. *J. Comput. Sys. Sciences*, 70:284–299, 2005.