Exploiting Problem Structure in Combinatorial Landscapes: A Case Study on Pure Mathematics Application

Xiao-Feng Xie
WIOMAX LLC, USA
xie@wiomax.com

Zun-Jing Wang
WIOMAX LLC, USA
wang@wiomax.com

Abstract

In this paper, we present a method using AI techniques to solve a case of pure mathematics applications for finding narrow admissible tuples. The original problem is formulated into a combinatorial optimization problem. In particular, we show how to exploit the local search structure to formulate the problem landscape for dramatic reductions in search space and for non-trivial elimination in search barriers, and then to realize intelligent search strategies for effectively escaping from local minima. Experimental results demonstrate that the proposed method is able to efficiently find best known solutions. This research sheds light on exploiting the local problem structure for an efficient search in combinatorial landscapes as an application of AI to a new problem domain.

1 Introduction

AI techniques have shown their advantages on solving different combinatorial optimization problems, such as satisfiability [Sutton *et al.*, 2010; Dubois and Dequen, 2001; Bjorner and Narodytska, 2015; Ansótegui *et al.*, 2015], traveling salesman problem [Zhang, 2004], graph coloring [Culberson and Gent, 2001], job shop scheduling [Watson *et al.*, 2003], and automated planning [Bonet and Geffner, 2001].

These problems can be generalized into the concept of combinatorial landscapes [Reidys and Stadler, 2002; Schiavinotto and Stützle, 2007; Tayarani-N and Prugel-Bennett, 2014], and problem solving can be cast to a search over a space of states. Such problems often are very hard [Cheeseman *et al.*, 1991]. In order to pursue an efficient search, it is vital to develop techniques to identify problem features and of exploiting the local search structure [Frank *et al.*, 1997; Hoffmann, 2001; Hoos and Stützle, 2004]. In particular, it is important to reduce and decompose the problem preserving structural features that enable heuristic search and with the effective problem size factored out [Slaney and Walsh, 2001; Mears and de la Banda, 2015]. It is also significant to tackle ruggedness [Billinger *et al.*, 2014] and neutrality (plateaus) [Collins, 2006; Benton *et al.*, 2010] in the landscapes.

In number theory, a k-tuple $\mathcal{H}_k$ is *admissible* if $\phi_p(\mathcal{H}_k) < p$ for every prime p, where $\mathcal{H}_k = (h_1, \ldots, h_k)$ is a strictly increasing sequence of integers, and $\phi_p(\mathcal{H})$ denotes the number of distinct residue classes modulo p occupied by the elements in $\mathcal{H}_k$ [Goldston *et al.*, 2009]. The objective is to minimize the *diameter* of $\mathcal{H}_k$, i.e., $d(\mathcal{H}_k) = h_k - h_1$, for a given k.

The early work [Hensley and Richards, 1974; Gordon and Rodemich, 1998; Clark and Jarvis, 2001] to compute narrow admissible tuples has been motivated by the incompatibility of the two long-standing Hardy-Littlewood conjectures.

Admissible sets have been used in the recent breakthrough work to find small gaps between primes. In [Goldston *et al.*, 2009], it was proved that any admissible $\mathcal{H}_{k_0}$ contains at least two primes infinitely often, if k_0 satisfies some arithmetic properties. In [Zhang, 2014], it was proved that a finite bound holds at $k_0 \geq 3.5 \times 10^6$. The bound was then quickly reduced to $d^*(\mathcal{H}_{105}) = 600$ [Maynard, 2015] and $d^*(\mathcal{H}_{50}) = 246$ [Polymath, 2014b], and wider ranges of k_0 also were obtained on bounded intervals containing many primes [Polymath, 2014b]. Moreover, admissible sets have been used to find large gaps between primes [Ford *et al.*, 2015].

Most of the existing techniques to find narrow admissible tuples are *sieve* methods [Hensley and Richards, 1974; Clark and Jarvis, 2001; Gordon and Rodemich, 1998; Polymath, 2014a; 2014b], although a few local optimizations were proposed recently [Polymath, 2014a; 2014b].

In this paper, we formally model this problem into a combinatorial optimization problem, and design search strategies to tackle the landscape, by utilizing the local search structure. Our solver is systematically tested to show its effectiveness.

2 Search Problem Formulation

For a given k, a *candidate number set* $\mathcal{V}$ with $|\mathcal{V}| \geq k$ can be precomputed, and a required *prime set* $\mathcal{P}$, where each prime $p \leq k$, can be determined. Each $\mathcal{H}$ is obtained by selecting the numbers from $\mathcal{V}$, and the admissibility is tested using $\mathcal{P}$.

Definition 1 (Constraint Optimization Model)**.** *For a given k, and given the required $\mathcal{V}$ and $\mathcal{P}$, the objective is to find a number set $\mathcal{H} \subseteq \mathcal{V}$ with the minimal $d(\mathcal{H})$ value, subject to the constraints $|\mathcal{H}| = k$ and $\mathcal{H}$ is admissible.*

For convenience, $\mathcal{V}$, $\mathcal{P}$, and $\mathcal{H}$ are assumed to be sorted in increasing order. We denote $\mathcal{H}$ as $\mathcal{H}_k$ if $|\mathcal{H}| = k$, as $\tilde{\mathcal{H}}$ if it is *admissible*, and as $\tilde{\mathcal{H}}_k$ if it satisfies both of the constraints.

Given $\mathcal{V}$ and $\mathcal{P} = (p_1, \cdots, p_i, \cdots, p_{|\mathcal{P}|})$, the following three data structures are defined for facilitating the search:

Definition 2 (Residue Array $\mathcal{R}_v$)**.** *For $v \in \mathcal{V}$, $\mathcal{R}_v$ is calculated on $\mathcal{P}$: its ith row is $r_{v,i} = v \bmod p_i$ for $i \in [1, |\mathcal{P}|]$.*

Definition 3 (Occupancy Matrix $\mathcal{M}$)**.** *$\mathcal{M}$ is an irregular matrix, in which each row i contains p_i elements corresponding to the residue classes modulo p_i. For any given number set $\mathcal{H} \subseteq \mathcal{V}$, there is $m_{i,j} = \sum_{v \in \mathcal{H}} \mathbb{1}(j \equiv r_{v,i} + 1)$ for $j \in [1, p_i]$, which means the count of numbers in $\mathcal{H}$ occupying each residue class modulo p_i.*

Definition 4 (Count Array $\mathcal{F}$)**.** *$\mathcal{F}$ is an array, in which each row i gives the count of zero elements in the ith row of $\mathcal{M}$.*

Property 1. *The space requirements for $\{\mathcal{R}_v | v \in \mathcal{V}\}$, $\mathcal{M}$, $\mathcal{F}$ are respectively $|\mathcal{V}| \cdot |\mathcal{P}|$, $\sum_{i \in [1,|\mathcal{P}|]} p_i$, and $|\mathcal{P}|$.*

Figure 1 gives the example for an admissible set $\tilde{\mathcal{H}}_7 = (0, 2, 8, 12, 14, 18, 30)$ with $d(\tilde{\mathcal{H}}_7) = 30$, the full prime set $\mathcal{P}$, $\mathcal{R}_v$ for each $v \in \tilde{\mathcal{H}}_7$, and the corresponding $\mathcal{M}$ and $\mathcal{F}$.

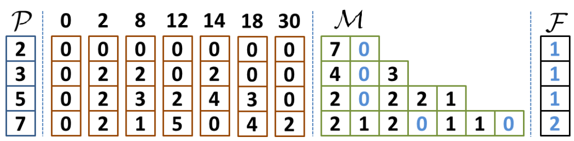


Figure 1: An admissible example for $\tilde{\mathcal{H}}_7$ with $d(\tilde{\mathcal{H}}_7) = 30$.

$\mathcal{H}$ and its corresponding $\mathcal{M}$ and $\mathcal{F}$ have a few properties:

Property 2 (Admissibility)**.** *$\mathcal{H}$ is admissible if $f_i > 0$, $\forall i$.*

There is a constraint violation at row i if $f_i = p_i - \phi_{p_i} \equiv 0$. The total *violation count* should be 0 for each $\tilde{\mathcal{H}}$.

Property 3. *Let $W_{i,j} = \{v \in \mathcal{H} | r_{v,i} \equiv j - 1\}$, it contains all numbers occupying (i, j) of $\mathcal{M}$, and there is $|W_{i,j}| = m_{i,j}$.*

Property 4. *For each row i, there is $\sum_{j \in [1,p_i]} m_{i,j} = |\mathcal{H}|$.*

There are two basic properties based on an admissible $\tilde{\mathcal{H}}_k$:

Property 5 (Offsetting)**.** *For any $c \in \mathbb{Z}$, $\mathcal{H}_k^{[c]} = (h_1 + c, \ldots, h_k + c)$ is admissible, and there is $d(\mathcal{H}_k^{[c]}) = d(\tilde{\mathcal{H}}_k)$.*

Property 6 (Subsetting)**.** *Any subset of $\tilde{\mathcal{H}}$ is admissible.*

Properties 5 and 6 were observed in [Polymath, 2014b]. Offsetting can be seen as rotating of residue classes at each row in $\mathcal{M}$; and subsetting does not decrease each row of $\mathcal{F}$.

Defining a compact $\mathcal{V}$ is nontrivial for reducing the size of problem space, which is exponential in $|\mathcal{V}|$.

One plausible way is to let $\mathcal{V}$ include all numbers in $[0, U]$, and set $h_1 = 0$. Let d_k^{LB} and d_k^{UB} be the best-so-far lower and upper bounds of the optimal value of $d(\tilde{\mathcal{H}}_k)$. During the search, $|\mathcal{V}| = U$ can be bounded by d_k^{UB}. However, it appears that d_k^{UB} is very close to $\lfloor k \log k + k \rfloor$ [Polymath, 2014b], which might still be very large when k is big.

Based on Property 4, as p_i is small, the average $m_{i,j}$ would be large. For the rows with $f_i = 1$, it would be difficult to find a useful heuristic for changing the unoccupied column j at row i. Intuitively, each unoccupied location (i, j) in $\mathcal{M}$ can be assumed to be unchanged during the search. Thus, *sieving* can be applied to remove any numbers in $[0, U]$ that occupy those unoccupied locations, which could

Algorithm 1 Obtain $\mathcal{V}$ and $\mathcal{P}$

Require: k, $[0, U]$

1: $\mathcal{P}_C = \{p \le k\}$; $\mathcal{P}_R = \{p < \sqrt{k \log k}\}$ // Let $p_m = \sqrt{k \log k}$
2: $\mathcal{V} = [0, U]$ sieving all v with $r_{v,i} \equiv 1$ for $p_i \in \mathcal{P}_R$
3: Obtain $\mathcal{M}$ and $\mathcal{F}$ for $\mathcal{H} = \mathcal{V}$ using $\mathcal{P} = \mathcal{P}_C$
4: $\mathcal{P}_L = \{p_i \in \mathcal{P}_C | f_i > 0\}$; $\mathcal{P} = \mathcal{P}_C - \mathcal{P}_L$
5: **return** $\mathcal{V}$, $\mathcal{P}$

be found using Property 3, for a set of the smallest primes $\mathcal{P}_R \in \mathcal{P}$. After sieving, the proportion of remaining numbers is $\alpha \approx 1 - \prod_{p_i \in \mathcal{P}_R}(1 - 1/p_i)$. The completeness of the problem space on the other combinations in $\mathcal{M}$ can be retained using a sufficiently large U, based on the principle behind Property 5, i.e., offsetting as a choice of residue classes.

Remark 1. *The original problem can be converted into a list of subproblems, where each subproblem takes each $v \in [0, U - d_k^{LB}] \cap \mathcal{V}$ as the starting point h_1 to obtain the minimal diameter for each $\tilde{\mathcal{H}}_k \subseteq \mathcal{V}_v = [v, v + d_k^{UB}] \cap \mathcal{V}$. The optimal solution is then the best solution among all subproblems.*

Decomposition [Friesen and Domingos, 2015] has been successfully used in AI for solving discrete problems. The new perspective of the problem has two features. First, each subproblem has a much smaller state space, as $|\mathcal{V}_v| \approx \alpha \cdot d_k^{UB}$. Second, for totally around $\alpha \cdot (U - d_k^{LB})$ subproblems, the good solutions of neighboring subproblems might share a large proportion of elements, providing a very useful heuristic clue for efficient adaptive search in this dimension.

Let $\mathcal{P}_\mathcal{C}$ contain all primes $p \le k$. In theory, $\mathcal{P} = \mathcal{P}_\mathcal{C}$, but we can reduce it to an effective subset. Based on Property 4, if p_i is large, the average $m_{i,j}$ would be small. Some rows of $\mathcal{F}$ would always have $f_i > 0$, even for $\mathcal{H} = \mathcal{V}$. The set of corresponding primes, named $\mathcal{P}_L$, thus can be removed from $\mathcal{P}$, without any loss of the completeness for testing the admissibility. The effective prime set would be $\mathcal{P} = \mathcal{P}_\mathcal{C} - \mathcal{P}_L$.

Algorithm 1 gives the specific realization for obtaining $\mathcal{V}$ and $\mathcal{P}$. Here $\mathcal{P}_R$ in Line 1 and $\mathcal{V}$ in Line 2 are obtained using the setting in the greedy sieving method [Polymath, 2014b].

3 Search Algorithm

In this section, the basic operations on auxiliary data structures are first introduced. Some search operators are then realized. Finally, we describe the overall search algorithm.

3.1 Operations on $\mathcal{R}_v$, $\mathcal{M}$ and $\mathcal{F}$

For every $v \in \mathcal{V}$, $\mathcal{R}_v$ is calculated in advance. For $\mathcal{H} \subseteq \mathcal{V}$, the corresponding $\mathcal{M}$ and $\mathcal{F}$ are synchronously updated locally. The space requirements are given in Property 1.

There are two elemental 1-*move* operators, i.e., *adding* $v \notin \mathcal{H}$ into $\mathcal{H}$ to obtain $\mathcal{H} = \mathcal{H} + \{v\}$, and *removing* $v \in \mathcal{H}$ from $\mathcal{H}$ to obtain $\mathcal{H} = \mathcal{H} - \{v\}$, for given $\mathcal{H} \subseteq \mathcal{V}$ and $v \in \mathcal{V}$.

Property 7 (Connectivity)**.** *The two elemental 1-move operators possess the connectivity property for each $\mathcal{H} \in \mathcal{V}$.*

The connectivity property [Nowicki and Smutnicki, 1996] states that there exists a finite sequence of such moves to achieve the optimum state from any state in the search space.

Algorithm 2 Update $\mathcal{M}$ and $\mathcal{F}$ as adding $v \notin \mathcal{H}$ into $\mathcal{H}$

Require: $R_v, \mathcal{M}, \mathcal{F}$
1: **for** $i \in [1, |\mathcal{P}|]$ **do**
2: $\quad j = r_{v,i} + 1$; $m_{i,j} = m_{i,j} + 1$
3: $\quad$ **if** $m_{i,j} \equiv 1$ **then** $f_i = f_i - 1$
4: **end for**
5: **return** $\mathcal{M}, \mathcal{F}$

Algorithm 3 Update $\mathcal{M}$ and $\mathcal{F}$ as removing $v \in \mathcal{H}$ from $\mathcal{H}$

Require: $R_v, \mathcal{M}, \mathcal{F}$
1: **for** $i \in [1, |\mathcal{P}|]$ **do**
2: $\quad j = r_{v,i} + 1$; $m_{i,j} = m_{i,j} - 1$
3: $\quad$ **if** $m_{i,j} \equiv 0$ **then** $f_i = f_i + 1$
4: **end for**
5: **return** $\mathcal{M}, \mathcal{F}$

For $\mathcal{H} \equiv \varnothing$, there are $m_{i,j} = 0$ for each i, j, $f_i = p_i$ for each i, based on Definitions 3 and 4. The $\mathcal{M}$ and $\mathcal{F}$ for any $\mathcal{H}$ can be constructed by adding each $v \in \mathcal{H}$ using Algorithm 2. For any two states $\mathcal{H}_A$ and $\mathcal{H}_B$, $\mathcal{H}_A$ can be changed into $\mathcal{H}_B$ by adding each $v \in \mathcal{H}_B - \mathcal{H}_A$ and by removing each $v \in \mathcal{H}_A - \mathcal{H}_B$. The total number of 1-moves is $L = |\mathcal{H}_A \cup \mathcal{H}_B| - |\mathcal{H}_A \cap \mathcal{H}_B|$, i.e., which can be seen as the *distance* [Reidys and Stadler, 2002] between two states. The shorter the distance, the more similar the two states are.

Algorithms 2 and 3 respectively give the operations of updating $\mathcal{M}$ and $\mathcal{F}$ for the two elemental 1-move operators.

Property 8 (Time Complexity)**.** *Algorithms 2 and 3 have the time complexity $O(|\mathcal{P}|)$ in updating $\mathcal{M}$ and $\mathcal{F}$.*

In the following realizations, we will focus on the search transitions between $\tilde{\mathcal{H}}$ states. The admissibility testing (Property 2) on each $\tilde{\mathcal{H}}$ is not explicitly applied. Instead, *VioCheck* in Algorithm 4 is used to check Δ, i.e., the violation count to be increased, if adding v into $\mathcal{H}$, using R_v, $\mathcal{M}$ and $\mathcal{F}$.

3.2 Search Operators

We first realize some elemental and advanced search operators to provide the transitions between admissible states.

Side Operators

Let $Side$={Left, Right} define the two mutually reverse sides of $\mathcal{H}$. For an admissible state $\tilde{\mathcal{H}}$, each side operator tries to add or remove a number at the given side of $\tilde{\mathcal{H}}$ to obtain the admissible tuple with a diameter as narrow as possible.

SideRemove just removes the element at the given $Side$ from $\tilde{\mathcal{H}}$, and its output is admissible, according to Property 6.

Algorithm 5 defines the operation *SideAdd* for adding a number v into $\tilde{\mathcal{H}}$. To retain the admissibility, the number to be added is tested using Algorithm 4 to ensure the admissibility.

Repair Operator

The *Repair* operator repairs $\tilde{\mathcal{H}}$ into $\tilde{\mathcal{H}}_k$ using side operators: While $|\tilde{\mathcal{H}}| < k$, the *SideAdd* operator is iteratively applied on each side of $\tilde{\mathcal{H}}$, and the better one is kept; While $|\tilde{\mathcal{H}}| > k$, the *SideRemove* operator is iteratively applied on each side of $\tilde{\mathcal{H}}$, and the better one is kept. Finally, $\tilde{\mathcal{H}}_k$ is obtained as $|\tilde{\mathcal{H}}| = k$.

Algorithm 4 *VioCheck*: Get the change of the violation count

Require: $v, \mathcal{H}$ // Include R_v and corresponding $\mathcal{M}$ and $\mathcal{F}$
1: $\Delta = 0$
2: **for** $i \in [1, |\mathcal{P}|]$ **do**
3: $\quad$ **if** $m_{i, r_{v,i}+1} \equiv 0$ **and** $f_i \equiv 1$ **then** $\Delta = \Delta + 1$
4: **end for**
5: **return** Δ // The change of the violation count

Algorithm 5 *SideAdd*: For adding a number into $\tilde{\mathcal{H}}$

Require: $\tilde{\mathcal{H}}, Side$ // Include $\{R_v\}$ and corresponding $\mathcal{M}$ and $\mathcal{F}$
1: **if** $Side \equiv$ Left **then** $v = h_1$ **else** $v = h_{|\tilde{\mathcal{H}}|}$ // Get side value
2: l = *GetIndex*$(v, \mathcal{V})$ // Obtain the index l of v in $\mathcal{V}$
3: **while** $l \in [1, |\mathcal{V}|]$ **do**
4: $\quad$ **if** $Side \equiv$ Left **then** $l = l - 1$ **else** $l = l + 1$
5: $\quad \Delta =$*VioCheck*$(v_l, \tilde{\mathcal{H}})$ // Use Algorithm 4 to add the lth number in $\mathcal{V}$
6: $\quad$ **if** $\Delta \equiv 0$ **return** $\tilde{\mathcal{H}} = \tilde{\mathcal{H}} \cup \{v_l\}$ // Ensure the admissibility
7: **end while**
8: **return** $\tilde{\mathcal{H}}$ // The original $\mathcal{H}$ is unchanged

Shift Search

Algorithm 6 gives the realization of the *ShiftSearch* operator. The side is selected at random (Line 2). Each shift [Polymath, 2014a] is realized by combining *SideRemove* and *SideAdd* (Line 4), leading to a distance of 2 to the original state. Starting from $\tilde{\mathcal{H}}_O$, we applies totally up to N_L shifts (Line 3) unless *SideAdd* fails (Line 5), and the best state is kept as $\tilde{\mathcal{H}}_N$ (Line 6). The state $\tilde{\mathcal{H}}_N$ is accepted immediately if $d_N \leq d_O$, or with a probability otherwise (Line 8), following the same principle as in simulated annealing [Kirkpatrick *et al.*, 1983].

Insert Moves

Algorithm 7 gives the realization of the *InsertMove* operator to work on the input $\tilde{\mathcal{H}}$ for obtaining an admissible output.

The operator is realized in three levels, as defined by the parameter $Level \in \{0, 1, 2\}$. For each v in a compact set $\mathcal{V}_{in} = [h_1, h_{|\tilde{\mathcal{H}}|}] \cap \mathcal{V} - \tilde{\mathcal{H}}$ (Line 2), the violation count Δ is calculated using Algorithm 4 (Line 3). At level 0, the value v is immediately inserted into $\tilde{\mathcal{H}}$ if $\Delta \equiv 0$ (Line 4). Otherwise, if $Level > 0$ and $\Delta \equiv 1$, the violation row i is found using *VioRow* (Line 5), which is simply realized by returning i as the conditions are satisfied at Line 3 of Algorithm 4, and then v is stored into the set Q_i (Line 5) starting from $\varnothing$ (Line 1).

At levels 1 and 2, we compare $|Q_i|$ and $m_{i,sb}$, where sb is the second best location in row i of $\mathcal{M}$. Based on Property 3, $|W_{i,sb}| = m_{i,sb}$. Note that the admissibility is retained after adding elements in Q_i and removing elements in $W_{i,sb}$.

Remark 2. *For $\tilde{\mathcal{H}}$=InsertMove($\tilde{\mathcal{H}}$), $d(\tilde{\mathcal{H}})$ is non-increasing at all levels. $|\tilde{\mathcal{H}}|$ is respectively increased by 1 and $|Q_i| - m_{i,sb}$ at levels 0 and 1, and keeps unchanged at level 2.*

In general, *InsertMove* is successful if it can increase $|\tilde{\mathcal{H}}|$. However, the neighborhood might contains too many infeasible moves, as many 1-moves would trigger multiple violations. It might be inefficient to use systematic *adjustments* [Polymath, 2014a]. Our implementation targets on feasible moves intelligently by utilizing the violation check clues.

Algorithm 6 *ShiftSearch*: Combine side moves on $\tilde{\mathcal{H}}$

Require: $\tilde{\mathcal{H}}_O$ // Parameter: $N_L \geq 1, \beta \geq 0$
$\tilde{\mathcal{H}} = \tilde{\mathcal{H}}_O$; $k_O = |\tilde{\mathcal{H}}|$; $d_O = d(\tilde{\mathcal{H}})$; $d_N = \infty$; $\tilde{\mathcal{H}}_N = \tilde{\mathcal{H}}$
$Side$=***RND***({Left, Right}) $RSide$ = Reverse of $Side$
for $l \in [1, N_L]$ **do**
  $\tilde{\mathcal{H}}$ =*SideRemove*($\tilde{\mathcal{H}}$, $RSide$); $\tilde{\mathcal{H}}$=*SideAdd*($\tilde{\mathcal{H}}$, $Side$)
  if $|\tilde{\mathcal{H}}| < k_O$ **break** // Stop search if *SideAdd* fails
  if $d(\tilde{\mathcal{H}}) < d_N$ **then** $d_N = d(\tilde{\mathcal{H}})$; $\tilde{\mathcal{H}}_N = \tilde{\mathcal{H}}$
end for
if $d_N \leq d_O$ **or** $\frac{0.5}{(d_N - d_O)^\beta}$ >*RND*(0, 1) **return** $\tilde{\mathcal{H}}_N$
return $\tilde{\mathcal{H}}_O$ // The original $\mathcal{H}$ is unchanged

Local Search

Algorithm 8 gives the realization of the *LocalSearch* operator. We will only focus on the case of improving the input state with $|\tilde{\mathcal{H}}| = k$. Let the input have $d_0 = d(\tilde{\mathcal{H}})$. The *SideRemove* operator is first applied for N_S times (Lines 1-3). Its output has $|\tilde{\mathcal{H}}| < k$, and $d_1 = d(\tilde{\mathcal{H}}) < d_0$. The *InsertMove* operator is then applied for up to N_I times (Lines 4-6). Based on Remark 1, the output has $d_2 = d(\tilde{\mathcal{H}}) \leq d_1$. If this step leads to $|\tilde{\mathcal{H}}| \geq k$, the final output after repairing definitely has a lower diameter than d_0. Otherwise, it is still possible to produce a better output as the state is being repaired (Line 7).

3.3 Region-based Adaptive Local Search (RALS)

The region-based adaptive local search (RALS) is realized to tackle the problem decomposition as described in Remark 1.

Let us consider the problem along the dimension of the numbers in $\mathcal{V}$. For each $\tilde{\mathcal{H}}_k$, it can be indexed by $(h_1, d(\tilde{\mathcal{H}}_k))$. Let $f^*(v)$ be the optimal diameter for all $\tilde{\mathcal{H}}_k$ with $h_1 = v$, we can form a set of points $\{(v_1, f^*(v_1)), \cdots, (v_{|\mathcal{V}|}, f^*(v_{|\mathcal{V}|})\}$. It can be seen as a one-dimensional fitness landscape representing the fitness function $f^*(v)$ on the discrete variable from $v \in \mathcal{V}$. Note that the optimal solution on this fitness landscape is the optimal solution of the original problem.

Essentially, we would like to focus the search on those promising regions where $f^*(v)$ has higher quality. Nevertheless, early search can provide some clues for narrowing down promising regions, even though the fitness landscape itself is not explicit at the beginning, as $f^*(v)$ at each v can be revealed through extensive local search.

Database Management

We use a simple database, denoted by DB, to keep the high-quality solutions $\tilde{\mathcal{H}}_k$ found during the search, and index each of them as $(v, f(v))$, where $v = h_1$, and $f(v) = d(\tilde{\mathcal{H}}_k)$ for each $\tilde{\mathcal{H}}_k$. For each v, only the best-so-far solution and the corresponding $f(v)$ is kept. Here $f(v)$ plays the role of a virtual fitness function that is updated during the search process to approximate the real fitness function $f^*(v)$.

The database is managed in a region-based mode. Specifically, the total range $[0, U]$ of the numbers in $\mathcal{V}$ is divided into N_R regions. There are three basic operations for DB.

The *DBInit* operator is used for providing the initialization. The greedy sieve [Polymath, 2014b] is applied to generate a state $\tilde{\mathcal{H}}_k$ in each region for forming the initial $f(v)$.

Algorithm 7 *InsertMove*: Local moves in $[h_1, h_{|\tilde{\mathcal{H}}|}]$ of $\tilde{\mathcal{H}}$

Require: $\tilde{\mathcal{H}}$ // Parameter: $Level \in \{0, 1, 2\}$
Initialize $\{Q_i = \varnothing | i \in [1, |\mathcal{P}|]\}$ // Use as $Level > 0$
for $v \in \mathcal{V}_{in} = [h_1, h_{|\tilde{\mathcal{H}}|}] \cap \mathcal{V} - \tilde{\mathcal{H}}$ **do**
  Δ =*VioCheck*(v, $\tilde{\mathcal{H}}$) // Algorithm 4
  if $\Delta \equiv 0$ **return** $\tilde{\mathcal{H}} = \tilde{\mathcal{H}} \cup \{v\}$ // Level 0: Insert one number
  if $\Delta \equiv 1$ **then** i = *VioRow*($v, \tilde{\mathcal{H}}$); $Q_i = Q_i \cup \{v\}$
end for
for $Level > 0$ **and** $i \in [1, |\mathcal{P}|]$ **do**
  if $|Q_i| > m_{i,sb}$ **return** $\tilde{\mathcal{H}} = \tilde{\mathcal{H}} + Q_i - W_{i,sb}$ // Level 1
end for
for $Level > 1$ **and** $i \in [1, |\mathcal{P}|]$ (In Random Order) **do**
  if $|Q_i| \equiv m_{i,sb} > 0$ **return** $\tilde{\mathcal{H}} = \tilde{\mathcal{H}} + Q_i - W_{i,sb}$
end for
return $\tilde{\mathcal{H}}$ // The original $\mathcal{H}$ is unchanged

Algorithm 8 *LocalSearch*: Remove & insert to improve $\tilde{\mathcal{H}}_k$

Require: $\tilde{\mathcal{H}}$, N_S, N_I // Parameters: $N_S \geq 1$, $N_I \geq 1$
1: **for** $n \in [1, N_S]$ **do**
2:   $Side$=*RND*({Left,Right}); $\tilde{\mathcal{H}}$=*SideRemove*($\tilde{\mathcal{H}}$, $Side$)
3: **end for**
4: **for** $n \in [1, N_I]$ **do**
5:   $\tilde{\mathcal{H}}$ =*InsertMove*($\tilde{\mathcal{H}}$); **if** $|\tilde{\mathcal{H}}| \geq k$ **break**
6: **end for**
7: **return** $\tilde{\mathcal{H}}_k$ =*Repair*($\tilde{\mathcal{H}}$)

The *DBSelect* operator is used for selecting one incumbent state to apply the search operation. In the region-based mode, there are two steps to provide the selection. In the first step, each region provides one candidate. In this paper, we greedily choose the best solution in each region. In the second step, the incumbent state is selected from the candidates provided by all regions. We consider the following implementation. At the probability γ, the candidate is selected at random. Otherwise, *tournament selection* is applied to select the best solution among totally N_T randomly chosen candidates.

The *DBSave* operator simply stores each $\tilde{\mathcal{H}}_k$ into DB, and updates $f(v)$ internally. Dominated solutions are discarded.

Algorithm Realization

Algorithm 9 gives the implementation of RALS to obtain $\mathcal{H}_k^*$ for a given k. First, $\mathcal{V}$ and $\mathcal{P}$ are initialized using Algorithm 1, using $U = \lceil 1.5 \cdot (k \log k + k) \rceil$ for the range $[0, U]$ (Line 1) to ensure $U > d_k^{UB}$. Afterward, the database DB with N_R regions is initiated using the *DBInit* operator (Line 2).

The search process runs T iterations in total. In each iteration, we first select one incumbent solution $\tilde{\mathcal{H}}_k$ from DB using the *DBSelect* operator (Line 4). Then the actual search tackles two parts of the problem. The *ShiftSearch* operator is used to search on the virtual fitness landscape $f(v)$ (Line 5). The *LocalSearch* operator is then applied to improve $f(v)$ locally (Lines 6-7). For each search operator in Lines 5-7, the *DBSave* operator is applied to store newly generated solutions. Finally, the best solution $\mathcal{H}^*$ in DB is returned.

Algorithm 9 RLAS algorithm to obtain $\mathcal{H}_k^*$ for a given k

```
Intialize V and P using Algorithm 1 // U = 1.5 · ⌈k log k + k⌉
DB=DBInit(N_R)                      // Initiate DB with N_R regions
for t ∈ [1,T] do
    H̃_k =DBSelect(DB)               // Select one incumbent solution from DB
    H̃_k =ShiftSearch(H̃_k); DBSave(H̃_k, DB)
    H̃_k =LocalSearch(H̃_k, 1, N_I1); DBSave(H̃_k, DB)
    H̃_k =LocalSearch(H̃_k, 2, N_I2); DBSave(H̃_k, DB)
end for
return H* in DB                     // Return the best solution stored in DB
```

4 Results and Discussion

We now turn to the empirical evaluation of the proposed algorithm. For the benchmark instances, we refer to an online database [Sutherland, 2015] that has been established and extensively updated to contain the narrowest admissible k-tuples known for all $k \leq 5000$. The algorithm is coded in Java, and our experiments were run on AMD 4.0GHz CPU. For each instance, 100 independent runs were performed.

4.1 Results by Existing Methods

Most existing techniques to solve this problem are constructive and sieve methods [Polymath, 2014a; 2014b]. The sieve methods are realized by sieving an integer interval of residue classes modulo primes $p < k$ and then selecting an admissible k-tuple from the survivors. The easiest way to construct a narrow $\tilde{\mathcal{H}}_k$ is using the first k primes past k [Zhang, 2014]. As an optimization, the sieve of Eratosthenes takes k consecutive primes, to search starting from $p < k$, in order to select one among the admissible tuples that minimize the diameter.

The Hensley-Richards sieve [Hensley and Richards, 1974] uses a heuristic algorithm to sieve the interval $[-x/2, x/2]$ to obtain $\tilde{\mathcal{H}}_k$, leading to the upper bound [Polymath, 2014b]:

$$H(k) \leq k \log k + k \log\log k - (1 + \log 2)k + o(k).$$

The Schinzel sieve, as also considered in [Gordon and Rodemich, 1998; Clark and Jarvis, 2001], sieves the odd rather than even numbers. In the shifted version [Polymath, 2014b], it sieves an interval $[s, s + x]$ of odd integers and multiples of odd primes $p \leq p_m$, where x is sufficient large to ensure at least k survivors, and m is sufficient large to ensure that the survivors form $\tilde{\mathcal{H}}$, $s \in [-x/2, x/2]$ is the starting point to choose for yielding the smallest final diameter.

As a further optimization, the shifted greedy sieve [Polymath, 2014b] begins as in the shifted Schinzel sieve, but the minimally occupied residue class are greedily chosen to sieve for primes $p > \tau\sqrt{k \log k}$, where τ is a constant. Empirically, it appears to achieve the bound [Polymath, 2014a]:

$$H(k) \leq k \log k + k + o(1).$$

Table 1 lists the upper bounds obtained by applying a set of existing techniques, including k primes past k, Eratosthenes (Zhang) sieve, Hensley-Richards sieve, Schinzel and Shifted Schinzel sieve, by running the code[1] provided in [Polymath, 2014b], on $k = \{1000, 2000, 3000, 4000, 5000\}$. The best known results are retrieved from [Sutherland, 2015].

[1] http://math.mit.edu/~drew/ompadm_v0.5.tar

Table 1: Upper bounds on $\mathcal{H}_k$ by existing sieve methods.

k	1000	2000	3000	4000	5000
k primes past k	8424	18386	28972	39660	50840
Eratosthenes	8212	17766	28008	38596	49578
Schinzel	8326	18126	28092	38418	49056
Hensley-Richards	8258	17726	27806	38498	48634
Shifted Schinzel	8190	17716	27500	37782	48282
Best known	7802	16978	26606	36610	46806

Table 2: Upper bounds on $\mathcal{H}_k$ by different RALS versions.

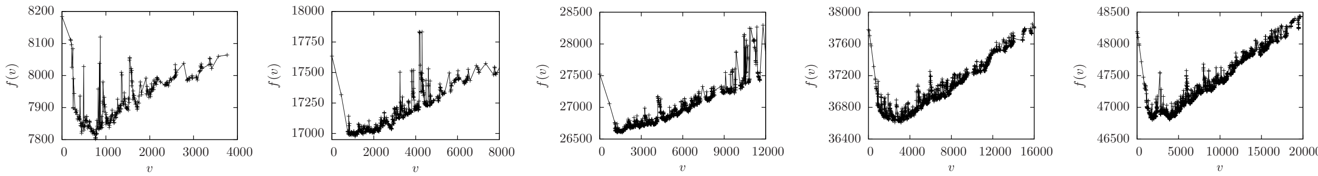
k	1000	2000	3000	4000	5000
BaseVer	7802.2	16981.6	26609.6	36626.2	46813.5
$T = 0$	7900.0	17204.0	26864.0	36926.0	47170.0
$Level = 0$	7835.3	17113.7	26797.5	36818.8	47060.3
$Level = 1$	7810.5	17055.0	26707.1	36742.7	46978.6
$N_{I1} = 100$	7802.5	16981.8	26613.0	36631.0	46815.0
$N_{I1} = 1000$	7802.1	16981.1	26608.1	36624.3	46813.6
$N_{I2} = 10$	7802.0	16980.5	26608.2	36626.4	46810.9
$\gamma = 0.001$	7802.3	16982.7	26613.4	36628.1	46818.3
$\gamma = 0.1$	7802.0	16979.0	26606.1	36623.2	46810.3
$\gamma = 1$	7803.2	16982.2	26607.7	36631.5	46814.2

4.2 Results by RALS algorithm

Table 2 lists the average results of different versions of the proposed RALS algorithm. The "BaseVer" version is defined with the following settings. For the database DB, we use $N_R = 20$. For its *DBSelect* operator, there are $\gamma = 0.01$ and $N_T = 4$. For the search loop, we consider $T = 1000$ iterations. For the *ShiftSearch* operator, we set $N_L = 10$ and $\beta = 1$. For the *InsertMove* operator, there is $Level = 2$. For the parameters of *LocalSearch* in Algorithm 9, we set $N_{I1} = 500$ and $N_{I2} = 0$. The other versions are then simply the "BaseVer" version with different parameters.

With $T = 0$, the algorithm returns the best results obtained by the shifted greedy sieve [Polymath, 2014a; 2014b] in the N_R regions. The results are significantly better than the sieve methods in Table 1. The search operators in RALS show their effectiveness as all RALS versions with $T > 0$ perform significantly better than the version with $T = 0$.

Note that "BaseVer" has $Level = 2$, we can compare the RALS versions with different levels $\{0, 1, 2\}$ in the *InsertMove* operator of *LocalSearch*. On the performance, the version with a higher level produces better results than that of a lower level. With greedy search only, the first *LocalSearch* works as an efficient *contraction* process [Polymath, 2014a]. As described in Remark 2, *InsertMove* performs greedy search at levels 0 and 1, but performs plateau moves at level 2, from the perspective of updating $|\tilde{\mathcal{H}}|$. At level 0, the search performs elemental 1-moves. At level 1, the search can be in a very large neighborhood although it has a low time complexity. Plateau moves is used at level 2 to find exits, as remaining feasible moves are more difficult to check. Finding exits to leave plateaus [Hoffmann, 2001; Frank *et al.*, 1997] has been an important research topic about the local search topology on many combinatorial problems [Bonet and Geffner, 2001; Benton *et al.*, 2010; Sutton *et al.*,



(a) $k = 1000$ (b) $k = 2000$ (c) $k = 3000$ (d) $k = 4000$ (e) $k = 5000$

Figure 2: Snapshot of the virtual fitness landscape $f(v)$, taking v as the start element of admissible k-tuples.

Table 3: Results of "BaseVer" with $\gamma = 0.1, N_{I2} = 10$.

(a) $T = 100$

k	1000	2000	3000	4000	5000
Average	7802.8	16981.9	26611.6	36633.4	46817.0
SuccRate(%)	79	46	49	0	7
Time (s)	14.7	40.7	83.3	147.6	267.8

(b) $T = 1000$

k	1000	2000	3000	4000	5000
Average	7802.0	16978.9	26606.2	36620.6	46809.4
SuccRate(%)	100	86	96	14	42
Time (s)	138.5	371.9	757.5	1415.1	2518.3

2010]. From the viewpoint of the *LocalSearch* operator, the plateau moves on the part solved by *InsertMove* help escaping from local mimina in the landscape of the subproblem.

We compare the RALS versions with different $N_{I1} \in \{100, 500, 1000\}$, as "BaseVer" has $N_{I1} = 500$. Especially for $k \in \{3000, 4000\}$, the improvements of using higher N_{I1} are extremely significant as N_{I1} increases from 100 to 500, but are less significant as N_{I1} further increases to 1000.

In "BaseVer", the second *LocalSearch* in Line 7 of Algorithm 9 is actually not used if it has $N_{I2} = 0$. As we increase N_{I2} to 10, the instance $k = 1000$ can be fully solved to the best known solution, and the instance $k = 5000$ can also be solved to obtain a significantly better result.

Lines 1-3 of Algorithm 8 might be viewed as perturbation, an effective operator in stochastic local search [Hoos and Stützle, 2004] to escape from local minima on rugged landscapes [Tayarani-N and Prugel-Bennett, 2014; Billinger *et al.*, 2014]. In RALS, the second *LocalSearch* essentially applies a larger perturbation than the first *LocalSearch*.

Table 2 also gives the comparison for RALS with $\gamma \in \{0.001, 0.01, 0.1, 1\}$ for selecting the incumbent state in *DB-Select*. The larger the γ, the more random the selection is. The best performance is achieved as $\gamma = 0.1$, neither too greedy nor too random. We can gain some insights from a typical snapshot of the virtual fitness landscape $f(v)$, as shown in Figure 2. It is easy to spot the valley with high-quality solutions, as they provide significant clues for adaptive search. Meanwhile, the noises on the fitness landscape might reduce the effectiveness of pure greedy search. Thus, there is a trade-off between greedy and random search.

Tables 3 lists the performance measures, including the average, the successful rate of finding best known solutions (SuccRate), and the calculating time, by the "BaseVer" version with both $\gamma = 0.1$ and $N_{I2} = 10$, as $T = 100$ and $T = 1000$. This version achieves high SuccRate for $k \in \{1000, 2000, 3000\}$, and moderate SuccRate for $k \in \{4000, 5000\}$, as $T = 1000$. It also reaches reasonable good SuccRate as $T = 100$, with a lower execution time.

Table 4: New upper bounds on $\mathcal{H}_k$ for $k \in [2500, 5000]$.

k	$\mathcal{H}_k^*$	δd	k	$\mathcal{H}_k^*$	δd	k	$\mathcal{H}_k^*$	δd
2547	22248	4	3407	30612	12	4167	38324	2
2548	22256	4	3408	30628	2	4168	38330	4
2736	24018	2	3409	30634	6	4169	38334	8
2737	24024	6	3410	30640	6	4170	38344	8
3026	26868	6	3411	30646	8	4171	38358	6
3357	30098	8	3412	30652	18	4614	42852	8
3358	30108	8	3413	30666	18	4615	42860	10
3374	30286	2	3414	30684	10	4634	43076	4
3375	30294	6	3415	30700	8	4809	44824	2
3376	30300	12	3424	30782	4	4810	44830	4
3377	30316	2	3473	31298	2	4860	45366	2
3378	30324	2	3474	31302	6	4861	45376	2
3379	30334	2	3475	31314	2	4928	46050	2
3404	30580	6	3487	31438	2	4929	46060	2
3405	30586	14	4107	37680	4	4956	46336	2
3406	30600	10	4108	37688	2	4957	46354	2

Finally, we apply RLAS to compare the results for $k \in [2500, 5000]$ in [Sutherland, 2015]. In Table 4, we list the new upper bound $\mathcal{H}_k^*$ and the improvement on the diameter δd for each k of the 48 instances. Eight instances among them have $\delta d \geq 10$. Thus, AI-based methods might make further contributions to pure mathematics applications.

5 Conclusions

We presented a region-based adaptive local search (RALS) method to solve a case of pure mathematics applications for finding narrow admissible tuples. We formulated the original problem into a combinatorial optimization problem. We showed how to exploit the local search structure to tackle the combinatorial landscape, and then to realize search strategies for adaptive search and for effective approaching to high-quality solutions. Experimental results demonstrated that the method can efficiently find best known and new solutions.

There are several aspects of this work that warrant further study. A deeper analysis might be applied to better identify properties of the local search topology on the landscape. One might also apply advanced AI strategies, e.g., algorithm portfolios [Gomes and Selman, 2001] and SMAC [Hutter *et al.*, 2011], to obtain an even greater computational advantage.

References

[Ansótegui *et al.*, 2015] C. Ansótegui, F. Didier, and J. Gabas. Exploiting the structure of unsatisfiable cores in MaxSAT. In *IJCAI*, pages 283–289, 2015.

[Benton *et al.*, 2010] J. Benton, K. Talamadupula, P. Eyerich, et al. G-value plateaus: A challenge for planning. In *ICAPS*, pages 259–262, 2010.

[Billinger *et al.*, 2014] S. Billinger, N. Stieglitz, and T. R. Schumacher. Search on rugged landscapes: An experimental study. *Organization Science*, 25(1):93–108, 2014.

[Bjorner and Narodytska, 2015] N. Bjorner and N. Narodytska. Maximum satisfiability using cores and correction sets. In *IJCAI*, pages 246–252, 2015.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *IJCAI*, pages 331–340, 1991.

[Clark and Jarvis, 2001] David Clark and Norman Jarvis. Dense admissible sequences. *Mathematics of Computation*, 70(236):1713–1718, 2001.

[Collins, 2006] M. Collins. Finding needles in haystacks is harder with neutrality. *Genetic Programming and Evolvable Machines*, 7(2):131–144, 2006.

[Culberson and Gent, 2001] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1):227–264, 2001.

[Dubois and Dequen, 2001] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *IJCAI*, pages 248–253, 2001.

[Ford *et al.*, 2015] K. Ford, J. Maynard, and T. Tao. Chains of large gaps between primes. *arXiv:1511.04468*, 2015.

[Frank *et al.*, 1997] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[Friesen and Domingos, 2015] A. L. Friesen and P. Domingos. Recursive decomposition for nonconvex optimization. In *IJCAI*, pages 253–259, 2015.

[Goldston *et al.*, 2009] D. A. Goldston, J. Pintz, and C. Y. Yíldírím. Primes in tuples I. *Annals of Mathematics*, 170(2):819–862, 2009.

[Gomes and Selman, 2001] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1):43–62, 2001.

[Gordon and Rodemich, 1998] D. Gordon and G. Rodemich. Dense admissible sets. In *International Symposium on Algorithmic Number Theory*, pages 216–225, 1998.

[Hensley and Richards, 1974] D. Hensley and I. Richards. Primes in intervals. *Acta Arithmetica*, 4(25):375–391, 1974.

[Hoffmann, 2001] J. Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *IJCAI*, pages 453–458, 2001.

[Hoos and Stützle, 2004] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier, 2004.

[Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, pages 507–523. 2011.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Maynard, 2015] J. Maynard. Small gaps between primes. *Annals of Mathematics*, 181(1):383–413, 2015.

[Mears and de la Banda, 2015] C. Mears and M. G. de la Banda. Towards automatic dominance breaking for constraint optimization problems. In *IJCAI*, pages 360–366, 2015.

[Nowicki and Smutnicki, 1996] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813, 1996.

[Polymath, 2014a] D. H. J. Polymath. New equidistribution estimates of Zhang type. *Algebra & Number Theory*, 8(9):2067–2199, 2014.

[Polymath, 2014b] D. H. J. Polymath. Variants of the selberg sieve, and bounded intervals containing many primes. *Research in the Mathematical Sciences*, 1(1):1–83, 2014.

[Reidys and Stadler, 2002] C. Reidys and P. Stadler. Combinatorial landscapes. *SIAM Review*, 44(1):3–54, 2002.

[Schiavinotto and Stützle, 2007] T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153, 2007.

[Slaney and Walsh, 2001] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *IJCAI*, pages 254–259, 2001.

[Sutherland, 2015] A. V. Sutherland. Narrow Admissible Tuples. http://math.mit.edu/~primegaps, 2015.

[Sutton *et al.*, 2010] A. Sutton, A. Howe, and L. Whitley. Directed plateau search for MAX-k-SAT. In *Annual Symposium on Combinatorial Search*, pages 90–97, 2010.

[Tayarani-N and Prugel-Bennett, 2014] M.-H. Tayarani-N and A. Prugel-Bennett. On the landscape of combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(3):420–434, 2014.

[Watson *et al.*, 2003] J.-P. Watson, J. Beck, A. Howe, and L. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2):189–217, 2003.

[Zhang, 2004] W. Zhang. Phase transitions and backbones of the asymmetric traveling salesman problem. *Journal of Artificial Intelligence Research*, 21:471–497, 2004.

[Zhang, 2014] Y. Zhang. Bounded gaps between primes. *Annals of Mathematics*, 179(3):1121–1174, 2014.