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

Xiao-Feng Xie, Zun-Jing Wang {xie,wang}@wiomax.com WIOMAX LLC

Summary

- Use AI techniques to find narrow admissible tuples, a case of pure mathematics applications
  - Formulate the original problem into a combinatorial optimization problem
  - Exploit the local search structure for reduction in search space & elimination in search barriers
  - Realize search strategies for tackling problem structure & escaping from local minima
- Shed light on exploiting the local problem structure for efficient search in combinatorial landscapes as an application of AI to a new problem domain

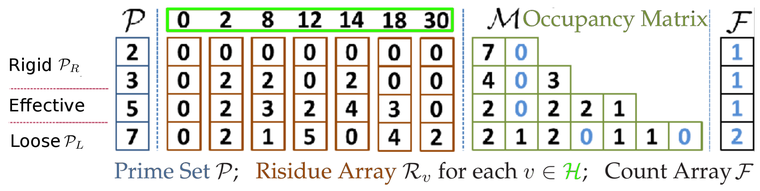
Basic Problem Formulation

In number theory, a k-tuple $\mathcal{H}_k = (h_1, \ldots, h_k)$ is *admissible* if $\phi_p(\mathcal{H}_k) < p$ for every prime p, where $\phi_p(\mathcal{H})$ denotes the number of distinct residue classes modulo p occupied by the elements in $\mathcal{H}_k$. 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.

- Motivated by the two long-standing H-L conjectures
- Recently used in finding bounded intervals containing multiple primes

[Constraint Optimization Model] For a given k, and the required number set $\mathcal{V}$ and prime set $\mathcal{P}$, 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 $\mathcal{H}_k$ is *admissible*.

[Auxiliary Data Definitions] $(\mathcal{H},\mathcal{P}) \to \{\mathcal{R}_v\} \to \mathcal{M} \to \mathcal{F}$, where $r_{v,i} = v \bmod p_i$, $m_{i,j}$ is the count of numbers in $\mathcal{H}$ occupying each residue class modulo p_i, f_i is the count of zero elements in the ith row of $\mathcal{M}$.



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

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

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

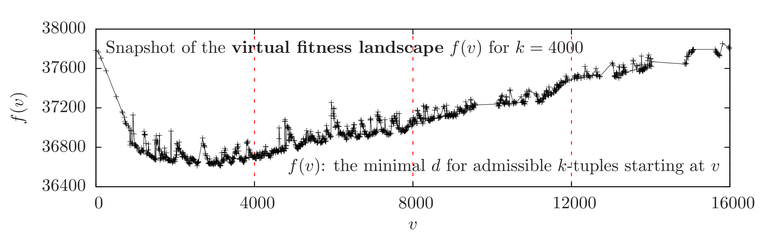
Reduction and Decomposition

[Problem Reduction] Obtain effective $\mathcal{V}$ and $\mathcal{P}$ (Alg. 1)

- Use the rigid set $\mathcal{P}_R$ to sieve the numbers in $\mathcal{V}$
- Use the remaining $\mathcal{V}$ to eliminate the loose set $\mathcal{P}_L$

[Problem Decomposition] Solve a list of subproblems

- Each subproblem takes each $v \in \mathcal{V}$ as the starting point h_1 to obtain the minimal diameter as $f^*(v)$
- Return the best result on the fitness landscape $f^*(v)$



Basic Operations

- Elemental 1-*moves*: adding/removing a number v
- *VioCheck*: Precheck the violation count as adding v
- Possess the *connectivity* property for each $\mathcal{H} \in \mathcal{V}$
- Each operation only costs $O(|\mathcal{P}|)$ in using $\mathcal{M}$ and $\mathcal{F}$
- The focus is on searching between admissible states

Search Algorithm

RALS Algorithm:

- Update the virtual fitness landscape $f(v)$ (as DB)
- Lines 4-5: Lateral search for a promising start v on f
- Lines 6-7: Extensive local search to obtain f^* near v

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

1: Intialize $\mathcal{V}$ and $\mathcal{P}$ using Algorithm 1 // $U = 1.5 \cdot \lceil k \log k + k \rceil$
2: DB=*DBInit*(N_R) // Initiate DB with N_R regions
3: **for** $t \in [1, T]$ **do**
4: $\tilde{\mathcal{H}}_k$ =*DBSelect*(DB) // Select one incumbent solution from DB
5: $\tilde{\mathcal{H}}_k$ =*ShiftSearch*($\tilde{\mathcal{H}}_k$); *DBSave*($\tilde{\mathcal{H}}_k$, DB)
6: $\tilde{\mathcal{H}}_k$ =*LocalSearch*($\tilde{\mathcal{H}}_k$, 1, N_{I1}); *DBSave*($\tilde{\mathcal{H}}_k$, DB)
7: $\tilde{\mathcal{H}}_k$ =*LocalSearch*($\tilde{\mathcal{H}}_k$, 2, N_{I2}); *DBSave*($\tilde{\mathcal{H}}_k$, DB)
8: **end for**
9: **return** $\mathcal{H}^*$ in DB // Return the best solution stored in DB

Lateral Search: Adaptive search in the rugged landscape

- *DBSelect*: Find a promising region, based on a mixed form of tournament selection and random selection
- *ShiftSearch*: Follow the acceptance style in *annealing*

Local Search: Efficiently search in a large neighborhood

- Target scarce feasible moves through $VioCheck$ info
- Make greedy search with one and more 1-*moves* (might be in a large neighborhood) at level 0 and 1
- Take plateau moves to find exits at level 2

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*($\tilde{\mathcal{H}}$)

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

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

Results

Results by Existing Methods: [Polymath, 2014a; 2014b]

- Most methods are constructive and sieve methods
- Empirically, the bound is $H(k) \leq k \log k + k + o(1)$
- Online database [Sutherland, 2015] for $k \leq 5000$

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

Results by RALS Algorithm: Different Parameters

- $T = 0$: The best results by the shifted greedy sieve
- $Level, N_{I1}, N_{I2}$: Local search with different levels, iterations, and contraction depths
- γ: The region selection with different randomness

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

Results by RALS Algorithm: Detail result statistics of "BaseVer" with $\gamma = 0.1, N_{I2} = 10$, with high SuccRate as $T = 1000$, and reasonable good as $T = 100$.

(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

Results by RALS Algorithm: New upper bounds for 48 instances, with 8 of them have $\delta d \geq 10$, as $k \in [2500, 5000]$.

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

Future Work

- Deeper analysis to identify more local search properties
- Toward general optimization with advanced AI strategies