Evolutionary n-level Hypergraph Partitioning with Adaptive Coarsening

"a tale of finding where EAs can contribute"
based on paper of same name, IEEE TEC 2019

Jim Smith and Richard Preen
Dept. Computer Science and Creative Technologies
University of the West of England

Why is this important?

- Because AI has been really successful at dealing with medium scale problems
- But now we're victims of our own success,

  So we're increasingly trying to optimize problems that are at or beyond memory/compute capacity so hybridization with mathematical solvers falls over*
- HyperGraph Partitioning is about splitting up huge problems, into balanced, minimally connected, sub units, that are computationally tractable
- It *complements* approaches to large-scale optimization

  like Divide-and-Conquer, cooperative coevolution, …
- Example application areas:
  - Economics: e.g. statistics for GDP, employment & trade,
  - Manufacturing: e.g. VLSI design,
  - Communications: adaptive network (re) configuration
  - Scientific computing in general.

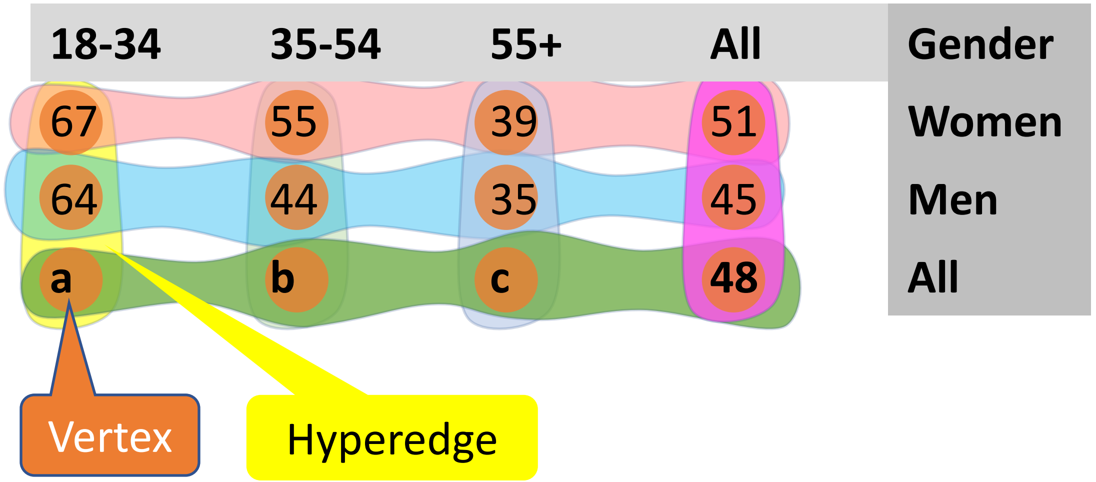
2

*e.g., Smith, et al . Genetic Approach to Statistical Disclosure Control. IEEE TEC, 16(3):431–441, 2012.

What's a hyper graph?

A generalization of a graph, where a **hyperedge** joins several vertices

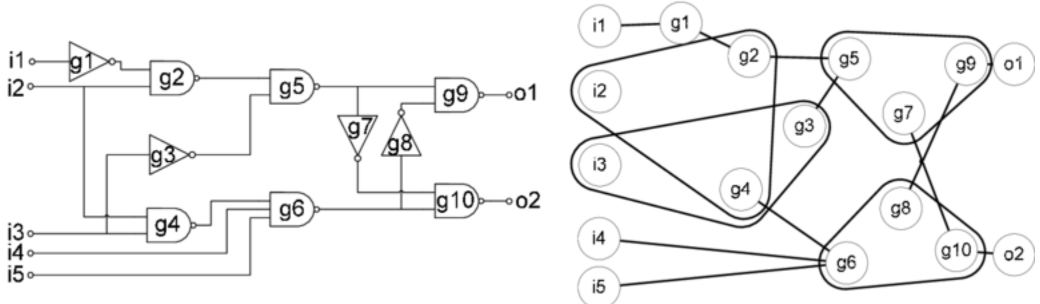
Simple example: statistical data
This just has 2 dimensions, with one level of hierarchy in each



Quickly becomes complex once you extend to multiple dimensions

UK Business Employment statistics has:
7 dimensions,
up to 6 levels of hierarchy
~10^8 cells, 10^5-10^6 hyperedges

This example has integers in the vertices,
Circuit diagrams have Booleans



Papa & Markov (2007) DOI: 10.1201/9781420010749.ch61

Hypergraph Partitioning

- Is about dividing the vertices into k approximately equal sets
- So as to minimize the number of cut hyperedges

- The combination of these constraints makes it NP-Hard

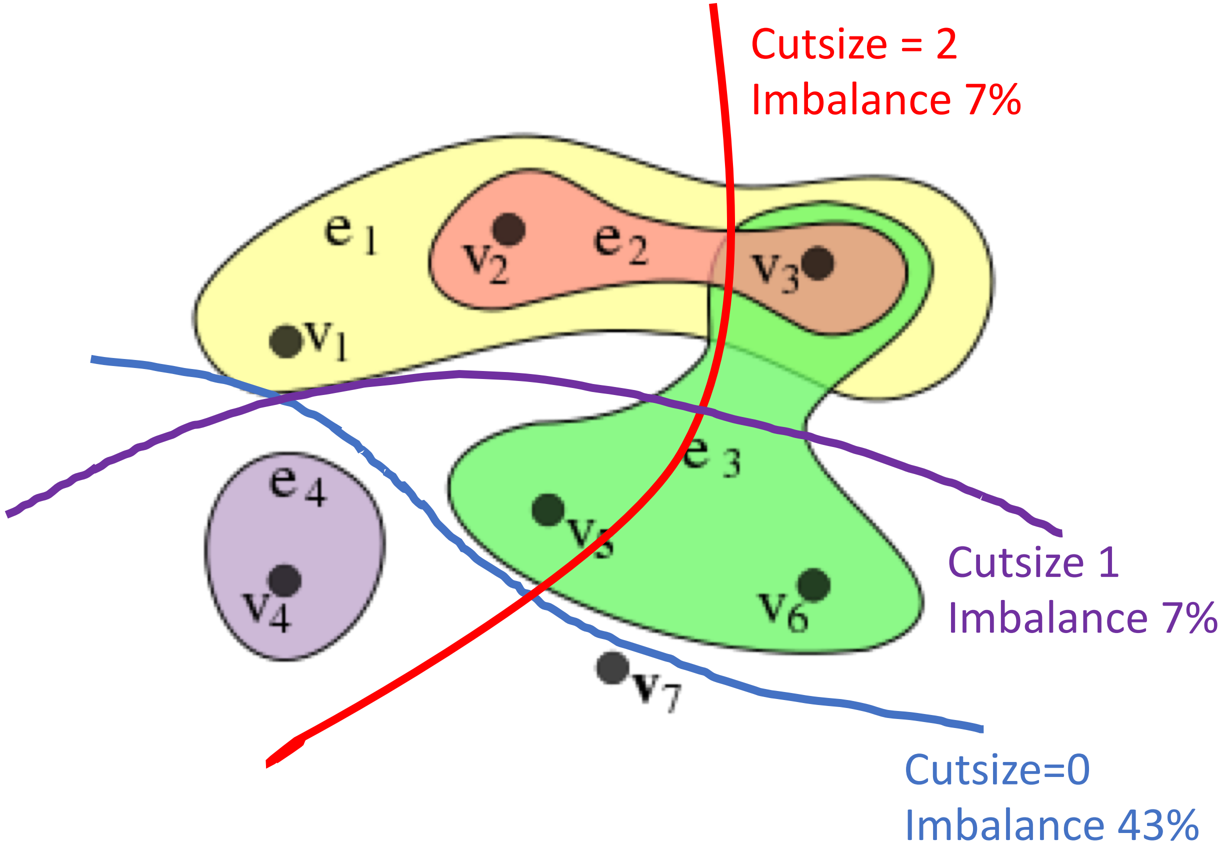
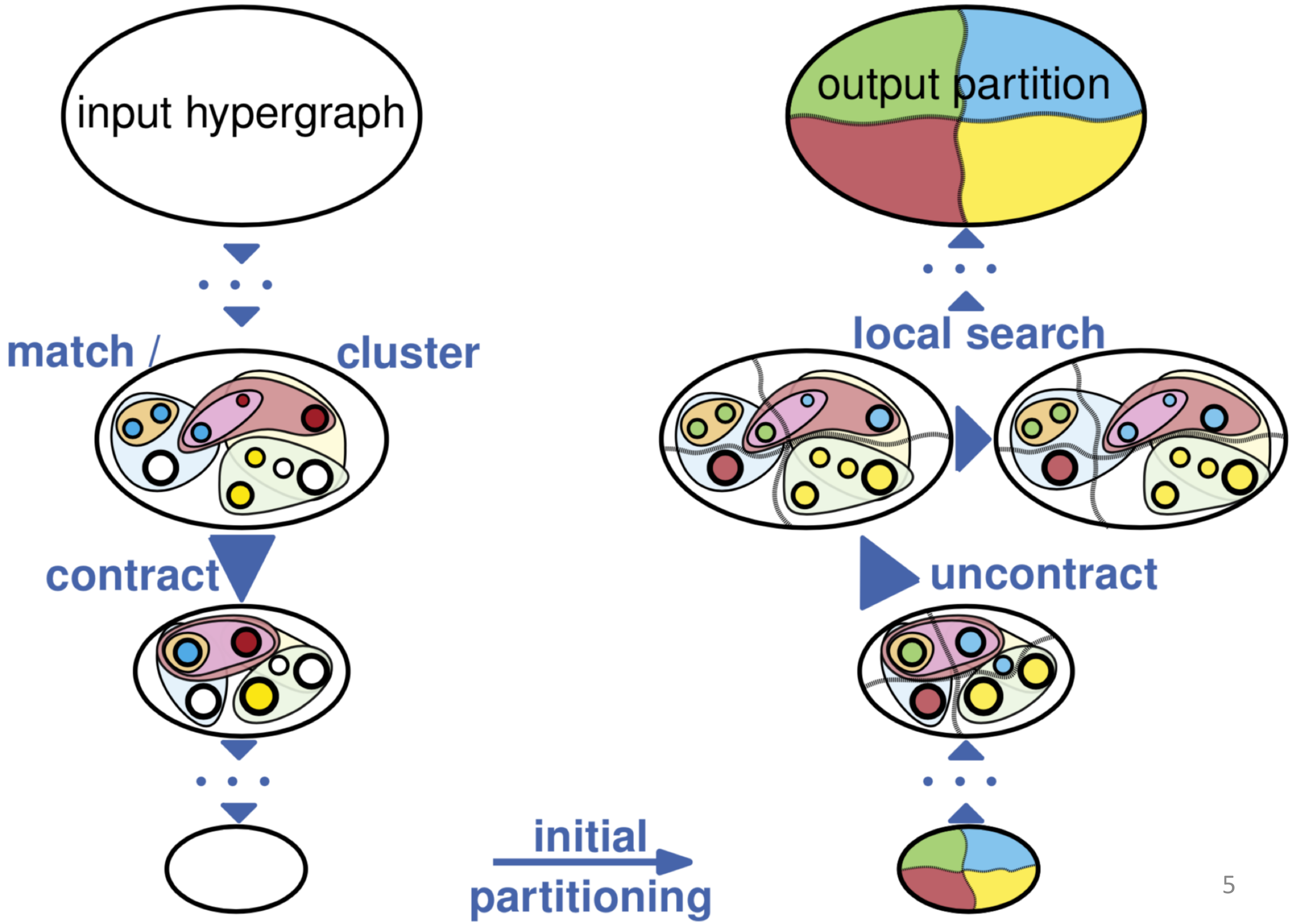


Image from https://tex.stackexchange.com/questions/1175/drawing-a-hypergraph

State of the art: Multi-level approaches

- If the original problems are too big to solve directly, so is the HGP!
- Classic Hypothesis: a good partitioning at one level is a good starting point for partitioning at the next.
- KaHyPar is currently s-o-t-a, and is open source



This image from : *Network Flow-Based Refinement for Multilevel Hypergraph Partitioning. Heuer et al.* SEA'18 .

Phase 1: understanding the problem

- Apply EAs & EDAs to do initial partitioning

  Build on , don't replicate other people's research effort in HGP or EAs for Graph Partitioning

- Initial Hypotheses:

  (1) We can exploit the symmetry of the problem within Estimation of Distribution Algorithms.

  (2) We can exploit self-adaptive mutation and use existing refiners within a Memetic Algorithm

Algorithm 2: Memetic EA($\mu + \lambda$) initial partitioner

$n = \frac{1}{|V|}$; $\mathcal{M} = [\frac{n}{100}, \frac{n}{10}, \frac{n}{5}, \frac{n}{2}, n, n, 2n, 5n, 10n, 100n]$
initialise parent population: $P = \{a_1 \dots a_\mu\}$
while *evaluation budget not exhausted* **do**
  /* create offspring population */
  for $i = 1$ *to* λ **do**
    parent $p_1 = RandomSelection(P)$
    parent $p_2 = RandomSelection(P)$
    offspring $a_i = p_1$
    if *rand()* $< \mathcal{X}$ **then**
      perform uniform crossover with normalised p_2
      if $p_1.fitness < p_2.fitness$ **then**
        $a_i.mut = p_2.mut$
      end
    end
    if *rand()* $< \mathcal{A}$ **then**
      $a_i.mut = RandomSelection(\mathcal{M})$
    end
    for *each hypernode in* a_i **do**
      if *drand()* $< a_i.mut$ **then**
        assign hypernode to a random partition
      end
    end
    repair partition if necessary
    apply FM local search (La
    evaluate a_i
  end
  /* select next parental population */
  $P = \mu$ fittest from $P + \lambda$
end

Addresses the 'competing conventions' problem

Self-adaptive mutation for combinatorial problems

Larmarkian constraint handling and local search

Phase 1: Initial Results

- Test set: 10 each from IBM VLSI circuits, U. Florida sparse matrices, 2014 SAT comp.
- Benchmark: against existing portfolio of 10 local search/ breadth-first methods with an even allocation of trials
- Standard KaHypar parameters extensively tuned by authors for bipartitioning,
- EA (global search) or EDA doesn't always do better that portfolio approach – overall not SSD on initial or final cut-size:

Why?

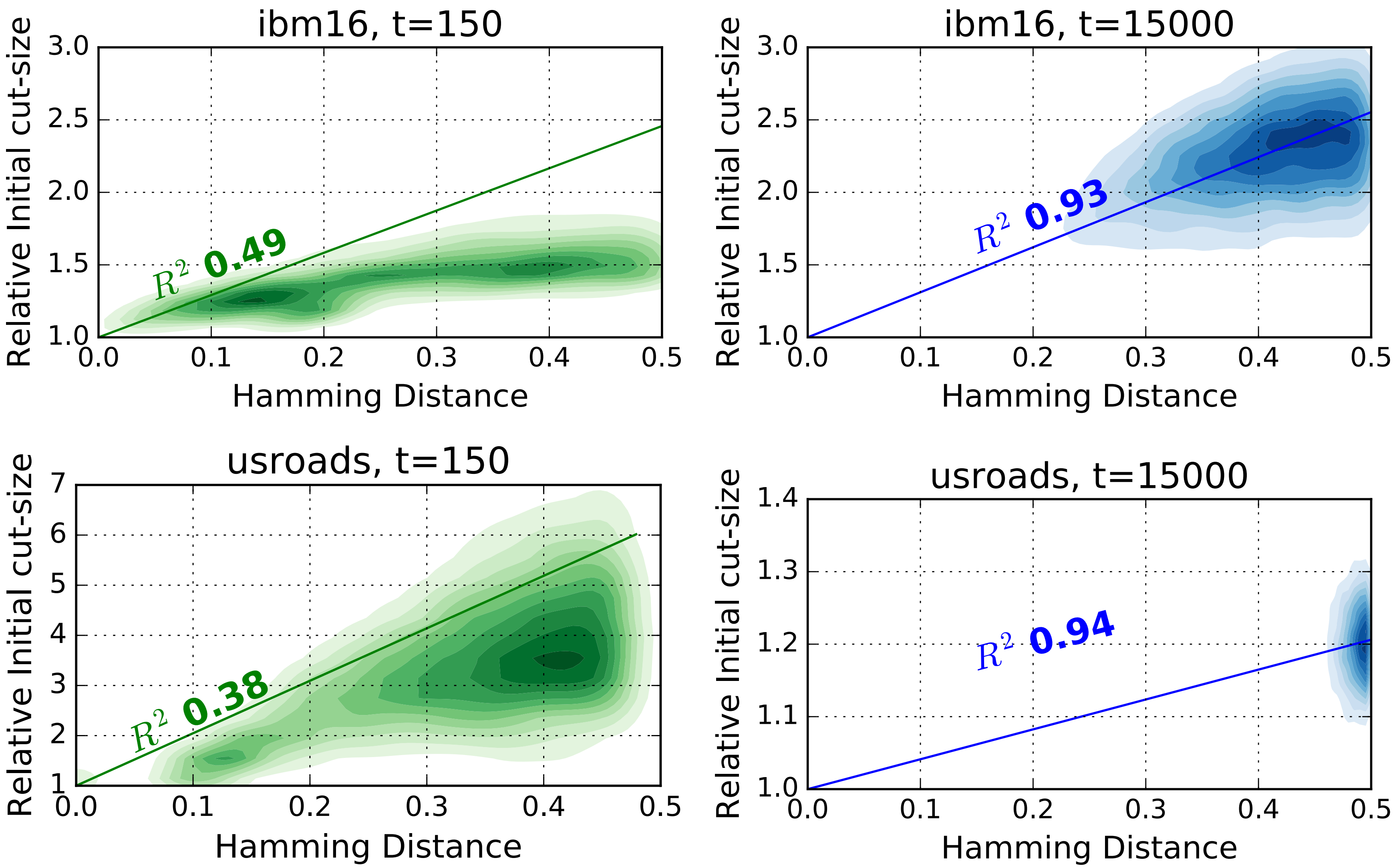
Phase 2: Understanding the problem landscapes

- KaHyPar coarsens until a threshold node limit is reached t_0*k
  - Typically in KaHyPar and other methods t_0 =150, so ~300 nodes
  - This makes the problem tractable for Breadth-first search etc.,
  - But what is the impact of the inevitable loss of information?

- We selected a 'training set' of 4 of each type of hypergraph then,
  - used KaHyPar to generate 10,000 local optima for hypergraphs
  - having stopped coarsening at t=150 and t=15000
  - measured distance of each solution to closest (estimated) local optimum, and relative solution quality

Kernel density plots:

- Y-axes chosen to show patterns, miss many poor optima for t=150
- Lines show linear regression and co-efficient of determination
- Note scatter plots were highly misleading



Results, meanings, implications for design choices

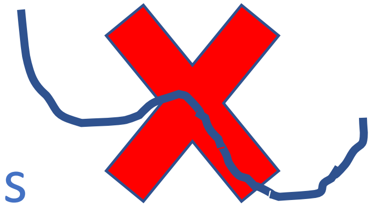
1. On some landscapes coarsening stopped prematurely around 40k nodes

   Algorithms should be able to cope with large search spaces

2. FM local search very effective, no correlation between cut-sizes before/after improvement, FDC low for t=150

   Lack of global structure: ‘good’ basins of attractions don’t have ‘good’ edges

   Algorithms should incorporate local search



3. Lots of distinct local optima: ~0 duplicates found, wide range of costs

   Worth devoting computational effort to good starting points for search

Results, meanings, implications for design choices

4. Positive Fitness-Distance Correlation on all landscapes
   Global optimum likely to be near other good local optima (‘big valley’)
   Suggests a role for population-based search with recombination
   This effect was *much* more noticeable at t=15000,
   Suggesting great role for EAs on these landscapes where there is less information loss

5. Big ‘gaps’ observed between best solution found and next
   Taken with lack of duplicates, suggests that there are barriers around the good solutions
   Lots of ‘next-best’ – concentric structure?
   (i) infeasibility of nearby solutions? (optima are likely to near ‘balance’ constraints),
   (ii) Large valley of attraction for 2nd best?
   Recombination and/or self-adaptation** to change role of mutation as search progresses

**Parameter Perturbation Mechanisms in Binary Coded GAs with Self-Adaptive Mutation. In *Foundations of Genetic Algorithms 7*, pp. 329–346, Morgan Kaufmann, San Francisco, 2003.

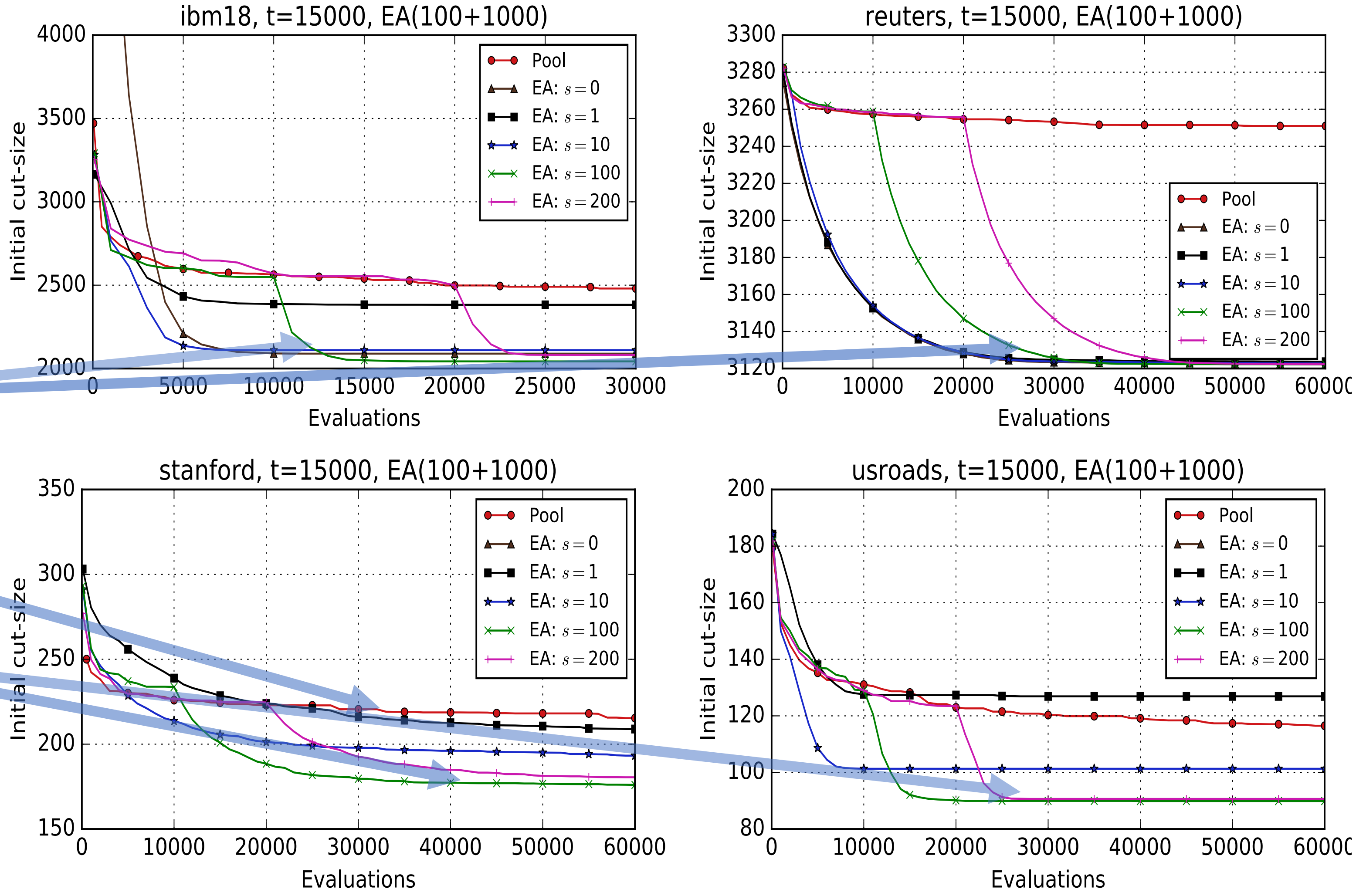
Verifying the design choices on the training set: Seeding

- μ = 100,λ = 1000
- Seed with best μ of s* μ calls to *Pool*
- *t=15000*

EA quickly discovers better solutions than pool for all s

s=0 too bad to fit on plot
S=1 no better than *Pool*,
S=100,200 SSD to others but not each other

In future we use s=100
i.e. 10,000 seed evaluations



Verifying the design choices on the training set: other EA parameters:

- Results confirmed our design choices for population management and variation operators were robust
  - (But we're not saying th ey couldn't be improved by a $2^{nd}/3^{rd}$ gen MA)

- EDA-based approaches failed

  - Univariate approaches gave poor results

  - Attempts to learn even simple pairwise models timed-out using different versions of Pelikan's BOA

Recap so far

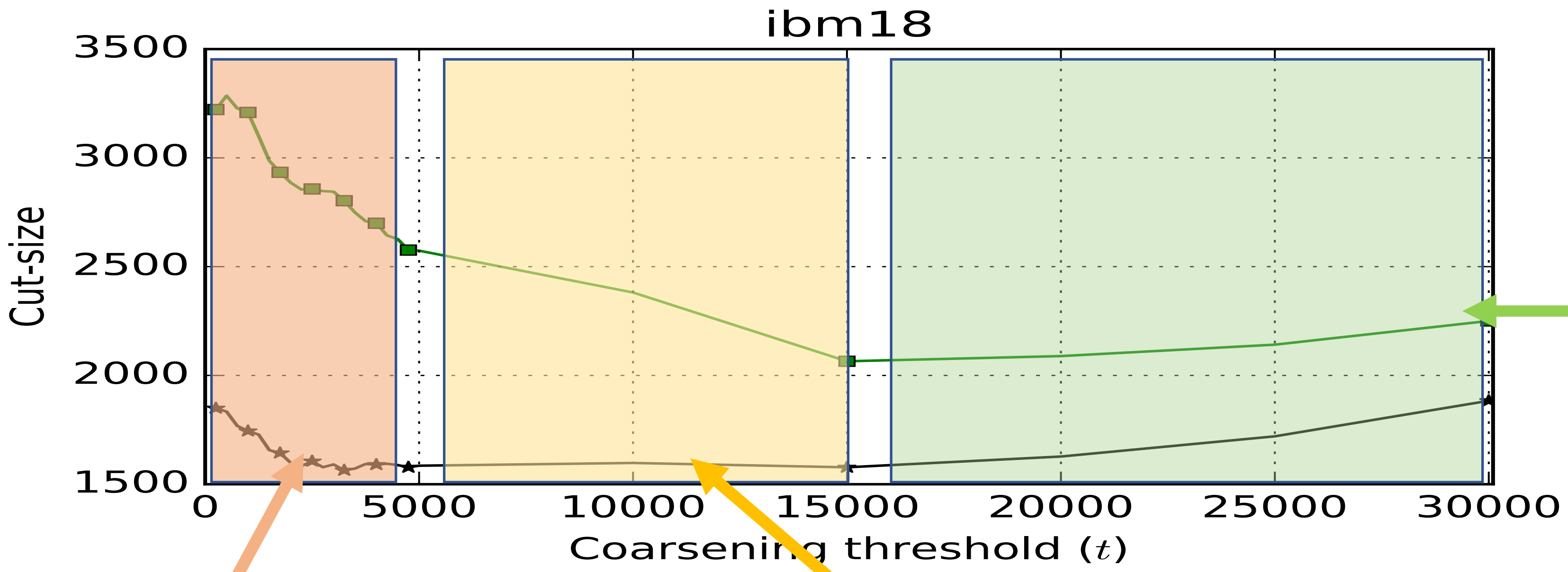
- Analysed existing approaches & identified initial partitioning as potentially fruitful area to apply insights from meta-heuristic search
- EA didn't provide expected gains over simple *Pool* algorithm at default thresholds
  - Could have stopped there and published this as a negative result,   instead
- Used landscape analysis to:
  - Understand the nature of the problem in general
  - Identify potential role for EA at less coarsened levels, when quality is most important
  - Make informed design decisions
- Verified design decisions using training set at t=15000:
  - Note high proportion of computational budget needed for seeding

But this is still at arbitrary threshold,

that takes no account of instance characteristics!

Phase 3: Finding optimal thresholds

- Selected training set of 12 HGs – 4 from each category
- Ran tests using EA and *Pool* with a number of different thresholds



Representative HG

Initial cutsize

Final cutsize

As you start to coarsen, the performance of the initial partitioning gets better and so does the final partitioning

At a final stage (t<3500) the FM can't save you and even the final partitioning gets worse

Eventually the initial partitioning starts to get worse because the space becomes increasingly complex and more rugged. Sophisticated uncoarsening retrieves similar final partitions

Phase 3: Effect of optimal thresholds on cut sizes

- Over all coarsening thresholds (AUC metric):
  EA significantly outperforms the Pool algorithm.

- Case-by-case: for all 12 hypergraphs

  EA final cut-sizes at t∗ are significantly smaller than the Pool algorithm at the default t=150.

- Across the 12: best-case cutsizes at t* for each alg.- instance combination:

  EA results are significantly better than the Pool algorithm (p ≤ 0.05).

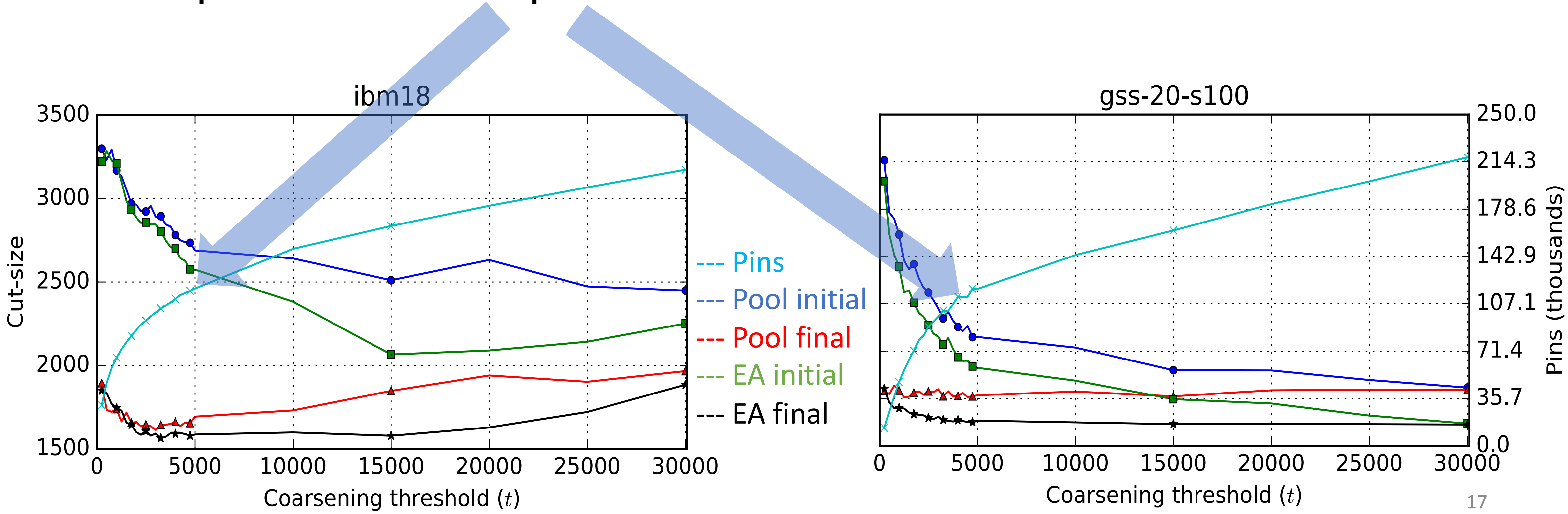
TABLE I

THE SMALLEST (AVERAGE) EA AND POOL FINAL CUT-SIZES ON FOUR HYPERGRAPHS FROM EACH OF THE BENCHMARK SETS AND THE RELATED COARSENING THRESHOLDS. CUT-SIZE HIGHLIGHTED IN BOLD FACE WHERE IT IS SIGNIFICANTLY DIFFERENT, $p \leq 0.05$.

Hypergraph	t^*_{Pool}	t^*_{EA}	cut^*_{Pool}	cut^*_{EA}	$\frac{time^*_{EA}}{time^*_{Pool}}$
ibm15	1000	3250	2649	2632	2.69
ibm16	3250	25000	1762	**1720**	3.15
ibm17	15000	15000	2276	**2244**	0.74
ibm18	3000	3250	1612	**1564**	0.57
Airfoil_2d	15000	15000	312	**311**	0.66
Reuters911	5000	10000	3199	**3125**	0.60
Stanford	500	250	30	29	0.40
usroads	750	2250	80	**79**	1.87
aaai10-planning	5000	5000	2312	**2261**	0.65
gss-20-s100	1250	30000	1002	**944**	9.67
MD5-28-2	500	10000	3580	**3483**	6.41
slp-synthesis	2500	4500	2618	**2549**	0.96

Phase 4: So how do we know where to stop?

- Based on the changing hypergraph characteristics not preset value
- Plotted changing number of *pins* (and other measures) during coarsening
- common pattern of ‘knee points’ as final and initial cutsize also deteriorated



Why do these occur?

- We hypothesize that change in performance is a result of information loss
  - Leading to more complex, rugged, unstructured landscapes for initial partitioning
  - And a loss of the relationship between quality of initial and final cutsizes

- We further hypothesize that the change in the reduction of pin count is (just one possible) proxy for this loss of information
  - Tends to decrease linearly to start with
  - Then there's a step-change as coarsening merges 'super-nodes'
    - which account for a lot of nodes,
    - and for a lot of differences between edges

Adaptive Stopping rule

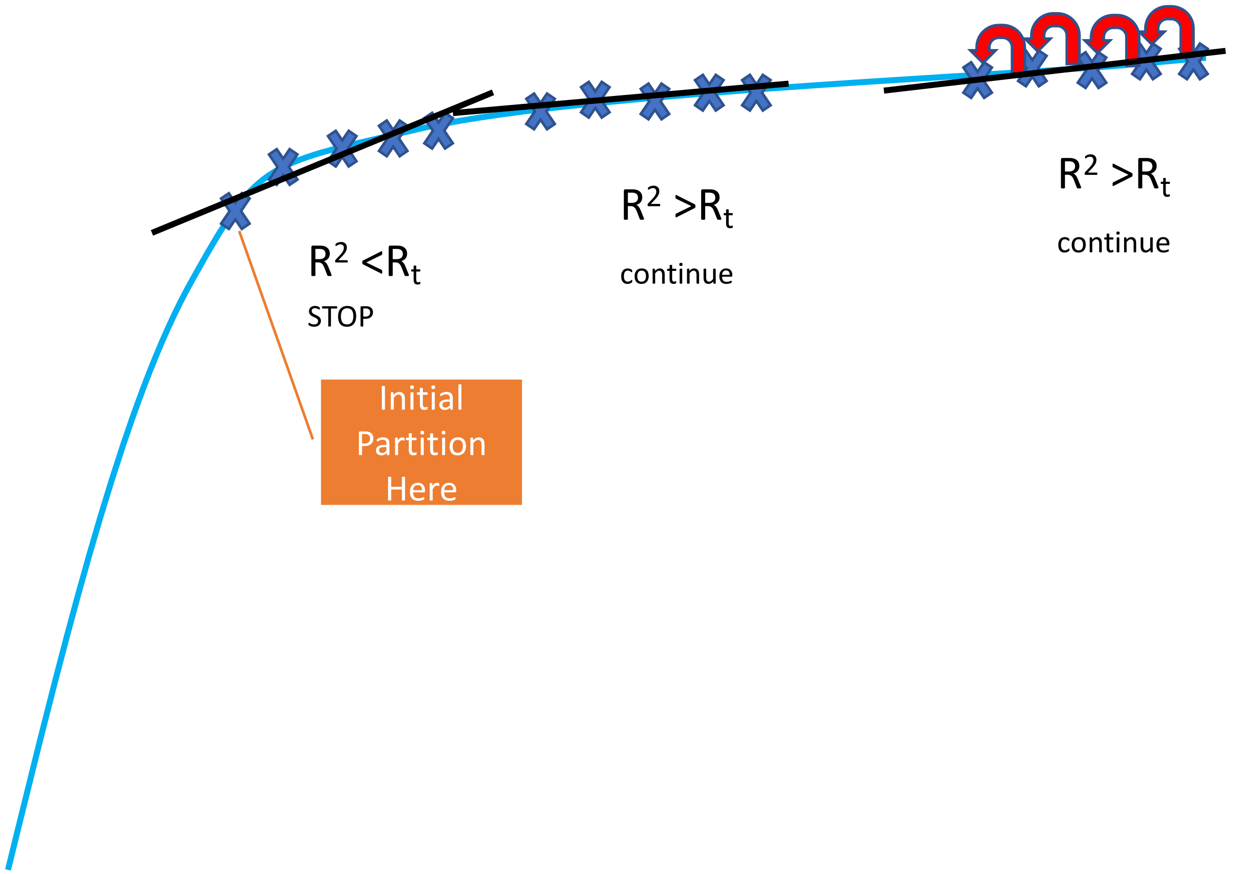
1. Take last *windowSize* values of pin_count
2. Perform Least Squares estimate of best-fit line
3. Calculate R^2
4. IF ($R^2 < R^2_{critical}$) OR ($t < t_{min}$):

   Stop and do initial partitioning
5. Else:

   Do *stride* uncoarsening steps

   Goto step 1

Tuned params via grid search over results from phase 3



Simple linear regression over sliding window as it is coarsened

Benchmarking Adaptive Stopping MA

focusing on 'mean final cut size' – the proof of the pudding!

1. **SSD Reductions in initial cut-size transfer to final cut size vs EA at t=150**
   - Across all 30 hypergraphs : Mean reduction of 1.6% ($p \leq 0.05$) vs. t=150; Best cut size reductions over 20%
   - Case-by-case: Mean final cut-size is smaller on 22 /30, SSD on 12/22 ($p<0.05$) Similar improvements vs. *Pool* at t=150.
2. **The stopping rule parameters generalise**
   - Across the 18 test hypergraphs: Overall reduction of 1.8% ($p \leq 0.05$) vs. EA at t=150.
   - Case by case: Mean final cut-size is smaller on 13 of the 18 hypergraphs. Cut sizes not SSD vs. t=15000, **But** the average wall-clock time was ≈ 7.4× faster. Vs t=15000
3. Total partitioning time: much faster (10X) at t=150 – but with larger cut size

Conclusions

- We have established a role for EA-based initial partitioning when solution quality is paramount
  - Complementing, other people's work
  - Evidence-based identification of role and design choices
- We have developed a new adaptive mechanism to stop coarsening based on the rate of change of information content
- This is a proof of concept - we're not claiming ours is the best MA for HGP - or that better rules don't exist

but we did beat the state of the art, sometimes by 20%

Future work

1. Lots of benchmarking and machine learning to determine:
   - More sophisticated adaptive stopping rules
   - Whether we can characterise Hypergraphs into different classes according measurements such as distributions of vertex degree, hyperedge size.

2. Improve the integration with the KaHyPar framework
   - reduce runtime
   - Look at other other niches for MAs and algorithm selection mechanisms

3. Apply to improve existing techniques at UK Office for National Statistics
   - Because the UK is really going to need accurate up-to-date information

Thanks to

- Xin Yao and SUSTech for the invitation and gracious hospitality
- Sebastian Schlag for providing access to KaHyPar and detailed discussions
- Martin Pelikan for making various versions of his BOA code available online
- You for listening

Tuning / robustness of stopping rule

Grid search over 12 hypergraphs in training set for which we had data from exhaustive search

- Size of the sliding window
- Stride of sliding window
- Critical value for R^2

looking for the set of values which predict the minima of the black line (final partition cost)

Results:

- window_size = 100, window_stride = 50,
- $R^2_{critical} = 0.99$, $t_{min} = 150$ (default)
- these may be be algorithm, and (of course) model, dependent