Design and analysis of heuristic algorithms

Marcus Ritt
Universidade Federal do Rio Grande do Sul
Porto Alegre

Mini-course ELAVIO 2013
September 2013
Contents

1 Introduction

2 Design of heuristics

3 Analysis of heuristics
What is the goal of research in heuristics?
What is the goal of research in heuristics?

- Being *better*?
What is the goal of research in heuristics?

- Being *better*?
- Being *faster*?
What is the goal of research in heuristics?

- Being *better*?
- Being *faster*?

Not only.
More goals

Often, we want also a

- **simple**, i.e. easy to implement, understand, and explain;
- **robust**, i.e. easy to calibrate, little sensible to parameters;
- **generalizable**, i.e. applicable to a large number of similar problems

heuristic (Barr et al. 1995; Cordeau et al. 2002).
Some possible questions

- *How* does a given heuristic $A$ behave?
  Study $A : I \rightarrow S$ & resource consumption (time, space, energy, . . .)
- Is there a *better* heuristic than $A$?
- What is the *best* heuristic for a given problem?
- *Why* does $A$ behave like it does?
Over the last decade and a half, tabu search algorithms for machine scheduling have gained a near-mythical reputation by consistently equaling or establishing state-of-the-art performance levels on a range of academic and real-world problems. Yet, despite these successes, remarkably little research has been devoted to developing an understanding of why tabu search is so effective on this problem class.
Despite widespread success, very little is known about why local search metaheuristics work so well and under what conditions. This situation is largely due to the fact that researchers typically focus on demonstrating, and not analyzing, algorithm performance. Most local search metaheuristics are developed in an ad hoc manner. A researcher devises a new search strategy or a modification to an existing strategy, typically arrived at via intuition. The algorithm is implemented, and the resulting performance is compared [...]

If the new algorithm outperforms existing algorithms, the results are published, advancing the state of the art. Unfortunately, most researchers [...] fail to actually prove that the proposed enhancements actually led to the observed performance increase (as typically, multiple new features are introduced simultaneously) or whether the increase was due to fine tuning of the algorithm or associated parameters, implementation tricks, flaws in the comparative methodology, or some other factors.
The field of optimization is perhaps unique in that natural or man-made processes completely unrelated to optimization can be used as inspiration, but other than that, what has caused the research field to shoot itself in the foot by allowing the wheel to be invented over and over again? Why is the field of metaheuristics so vulnerable to this pull in an unscientific direction? The field has shifted from a situation in which metaheuristics are used as inspiration to one in which they are used as justification, a shift that has far-reaching negative consequences on its credibility as a research area.
The field’s *fetish with novelty* is certainly a likely cause. [...] A second reason for this research to pass is the fact that the research literature in metaheuristics is positively obsessed with playing the up-the-wall game (Burke et al., 2009). There are no rules in this game, just a goal, which is to *get higher up the wall* (which translates to “obtain better results”) than your opponents. Science, however, is not a game. Although some competition between researchers or research groups can certainly stimulate innovation, the ultimate *goal of science is to understand*. True innovation in metaheuristics research therefore does not come from yet another method that performs better than its competitors, certainly if [it] is not well understood why exactly this method performs well.
Immediate consequences

- *Why* matters more than *how* or *that*: performance needs explanation.
- The contribution of *individual components* should be analyzed.
- Tuning of the algorithm should be quantified: *complexity* of separate steps & empirical complexity.
Compare: Scientific approach

- **Measure & describe**: Find a law.
  
  Time to fall to the ground: $\sqrt{2s/g}$.

- **Explain**: Reduce a phenomenon to a more abstract concept (greater *reach*!)
  
  Masses exert gravitational force on other masses.

- **Innovate**: Find a new abstract concept.
  
  General relativity: mass curves space.
Contents

1 Introduction

2 Design of heuristics
   • Design principles
   • Case study

3 Analysis of heuristics
A glimpse on algorithm engineering
A more detailed view
A general conclusion

i) Start with a well-defined, clear *scientific question*;

ii) generate several *hypotheses* to answer that question;

iii) devise *experimental tests* to support or reject (statistically) the *predictions* of the hypotheses;

iv) analyze the experimental results and conclude; this may generate new hypotheses.
What matters most in heuristics?

In order of importance (Watson et al. 2006; Hertz et al. 2003):

- Problem-specific techniques;
- the heuristic (most important: overcome stagnation);
- intensification and diversification as general beneficial strategies;
- the parameter settings;
- an efficient implementation.

(Inversions are possible.)
In general a *component-based approach* seems most adequate.

1. **Problem representation.**

   - Study different problem *representations*.
   - Project an adequate, efficient data structure for the principal operations (adding, deleting, changing elements, incremental evaluation) & analyze their complexity.
   - Consider to reduce the problem to a *heuristic core*.
   - Try to identify the *most difficult subproblem* that can be solved in polynomial time.
A simple example: SALBP
A simple example: SALBP

SALBP-1: For \( m = 6 \): \( c = 40 \) minimal.
A simple example: SALBP

SALBP-2: For $c = 49$: $m = 4$ minimal.

$T_1 = 46$
$T_2 = 49$
$T_3 = 46$
A simple example: SALBP

- Simple representation assigning tasks to stations *often invalid*.
- Alternative: *random keys* define a task order.
- Extract global order by *topological sort* breaking ties by random keys.
- For a given task order the residual problem is solvable by dynamic programming in $O(nm)$.
- The optimal cycle time $C(i, k)$ for tasks $i, \ldots, n$ on $k$ machines satisfies

$$C(i, k) = \begin{cases} \min_{i \leq j \leq n} \max \{\sum_{i \leq j' \leq j} t_{j'}, C(j + 1, k + 1)\} & i \leq n, k > 0 \\ 0 & i > n \\ \infty & i \leq n, k = 0 \end{cases}$$
2. Basic operations

- Propose *several operations* for construction, modification, and recombination of solutions.
- *Evaluate* them (statistically).
- For modification operators: consider *reduced* or *very large* neighborhoods.
Component-based design (cont.)

3. Basic heuristic

- **Combine basic operations** systematically to design a simple heuristic which overcomes local minima or a constructive heuristic.

- Conduct specific experiments to test whether techniques for overcoming local minima are effective.

- **Evaluate** the contribution of the individual components and their interactions.

- Proceed from simpler to more advanced techniques (e.g. local search, Simulated Annealing, Tabu Search; or greedy construction, bubble search, ACO).
4. Refined heuristic

- Add strategies for *intensification and diversification* using long-term memory.
- Proceed from simpler to more advanced techniques (e.g. Probe, GRASP-PR, genetic algorithm, scatter search).
Some more conclusions

- Compare during the design with the state-of-the-art in exact, approximate and heuristic algorithms.
- Search for explanations of the behavior of the heuristic.
- Experiments have to be reproducible. Bite the bullet: publish code (Barr et al. 1995; LeVeque 2013).
Example case study

Job-shop scheduling $J || C_{\text{max}}$:

- Machines $M_1, \ldots, M_m$.
- Jobs $J_1, \ldots, J_n$ each with some processing order for the machines.
- Job $j$ on machine $i$ has processing time $p_{ij}$.
- No preemption, no parallelism.
Nowicki and Smutnicki’s i-TSAB

- Tabu search TSAB using N5 modification operator.
- On stagnation or iteration limit restarts from some previous best solution a limited number of times (”back jump tracking”).
- Strategic intensification around good solutions.
- Strategic diversification by path relinking.

Despite its effectiveness, almost nothing is known about how both these and secondary components interact, [...] or even if all of the components are necessary. Watson et al. (2006)
N5 modification operator

- $C_{\text{max}}$ is determined by the critical path length.
- Critical path decomposable into blocks of jobs on the same machine.
- Transposing operations can shorten critical path only at block borders.
- N5 transposes
  - Last two operations of first and first two operations of last block.
  - First and last two operations of "inner" blocks.
Intensification and diversification

- Initial phase constructs pool of *elite solutions*. New elite solution results from TSAB applied to midpoint between seed solution and last elite solution.

- Work phase *repeatedly applies TSAB* to midpoint between incumbent and most distant elite solution $S$, and replaces $S$.

- i-TSAB terminates when diameter of elite set too small.
Analysis of Watson et al. (2006)

Deconstructing Nowicki and Smutnicki’s i-TSAB tabu search algorithm for the job-shop scheduling problem

- What is the influence of the core metaheuristic?
- What is the influence of intensification and diversification?
- How to balance intensification and diversification best?
Trap detector CMF

- Initial solution $s$, heuristic $H$, minimum distance $d_{\text{min}}$, distances $L_0$ e $\Delta L$ and interval $t_{\text{test}}$.

```plaintext
CMF(s) :=

  $s_t := s$

  each $t_{\text{test}}$ iterations:

  if $d(s, s_t) < d_{\text{min}}$ then
    if escaping then $L := L + \Delta L$ else $L := L_0$
    $s_t := s$
    $s := \text{randomWalk}(s, L)$
    escaping := true
  else
    $s_t := s$
    escaping := false
  end if

  $s := N_H(s)$ \{ heuristic step \}

end
```
Analyzing the core metaheuristic

- Comparison of Simple Tabu search, iterated local search, rejectionless Metropolis algorithm.
- Embedded into *trap detector* CMF.
- Principal findings:
  - Very similar results, Tabu search slightly better.
  - Strength comes likely from N5 modification operator.
  - i-TSAB performs still better: need for intensification & diversification.
Intensification and diversification

- Base heuristic $H(x, i)$ runs on $x$ for $i$ steps.
- Heuristic $H^*$ repeats $H$ until stagnation.

1. $H^*(x_0) :=$
2. $x := H(x_0, i_0)$
3. while $\varphi(x) < \varphi(x_0)$
4.     $x_0 := x$
5.     $x := H(x_0, i_1)$
6. end
7. return $x_0$
8. end
Generic intensification and diversification metaheuristic framework (IDMF)

1. \( \text{IDMF}() := \) 
2. generate population \( E \) of local optima 
3. apply \( H^*(e) \) to every \( e \in E \) 
4. repeat 
   5. with probability \( p_i \): \{ intensify \} 
      a. select \( e \in E \) 
      b. \( g := e \) 
   6. with probability \( 1 - p_i \): \{ diversify \} 
      a. select \( e, f \in E \) 
      b. generate midpoint \( g \) between \( e \) and \( f \) by path relinking 
      c. \( e' := H^*(g) \) 
      d. if \( \varphi(e') < \varphi(e) \) 
         e. \( e := e' \) 
5. end
Analyzing intensification/diversification

- Same algorithms: Simple Tabu search, iterated local search, Metropolis algorithm.
- Embedded in trap detector CMF.
- Additionally embedded into intensification/diversification framework IDMF.
- Principal findings:
  - IDMF always improves performance, often statistically significant.
  - Both intensification and diversification necessary, equal frequency ($p_i = 0.5$) tends to be best.
  - Tabu search continues to have a small advantage, and only slightly underperforms i-TSAB.
“... state-of-the-art performance depends primarily on the presence of two key algorithmic features: the N5 operator and a balance of straightforward intensification and diversification”
Contents

1 Introduction

2 Design of heuristics

3 Analysis of heuristics
   • Theoretical analysis
   • Practical analysis
   • Some practical tools
   • Case study
When is an algorithm efficient?

A first approach:

**Definition 3.1 (Kleinberg and Tardos (2005))**

An algorithm is efficient if, when implemented, it runs quickly on real input instances.
Contents

1 Introduction

2 Design of heuristics

3 Analysis of heuristics
   • Theoretical analysis
   • Practical analysis
   • Some practical tools
   • Case study
Theoretical approach

- Standard model: *RAM* (random access machine). Components:
  - processor, some registers, instruction pointer,
  - infinite memory of integers,
  - elementary instructions (control, transfer w/ indirect addressing, arithmetic).

- Algorithm complexity depends on instance size $n$. Interest: *Growth* with $n$.

- Constant factors & low order terms disregarded.
1. Introduction

2. Design of heuristics

3. Analysis of heuristics
   - Theoretical analysis
   - Practical analysis
   - Some practical tools
   - Case study
Experimental approach

- Measure on large number of instances, extrapolate, compare.
- Measure on selected instances and compare.

Good:
- Simple.

Bad:
- Can be a huge effort.
- Lots of dependencies (hardware, compiler, cache, I/O, noise, etc.) difficult comparison.
- Choice of representative instances non-trivial.
Some instance libraries

QAPLIB

Bin Packing

SteinLib

VRP web

VRPLIB

SATLIB

CSPLib

MIPLIB
Theoretical complexity

- For heuristics: often best to separate into
  - Polynomial part: complexity of an construction, iteration, etc.
  - Dynamic part: \# of overall iterations.
- Allows us to analyze implementation aspects (algorithms, data structures) separately.
Contents

1 Introduction

2 Design of heuristics

3 Analysis of heuristics
   - Theoretical analysis
   - Practical analysis
   - Some practical tools
   - Case study
Empirical complexity

- Time complexity of an algorithm is whp

\[ T(n) \sim ab^n n^c \log^d n \]

(Sedgewick and Wayne 2011; Sedgewick 2010).

- Two frequently reasonable hypotheses:
  - exponential For \( T(n) \sim ab^n \) we have
    \[ \log T \sim \log a + n \log b \]
  - polynomial For \( T(n) \sim an^b \) we have
    \[ \log T \sim \log a + b \log n. \]

- In both cases: coefficients by linear regression.
Example: Permutation flow shop

- Heuristic of Nawaz et al. (1983).
  - Order jobs in non-increasing order of total duration.
  - Greedily insert at best position in current sequence.
- Naively: $O(n^3m)$, Taillard (1990) gave $O(n^2m)$ algorithm.
- Experimental data from Rad et al. (2009).
- Polynomial model $\log T \sim a \log n + b \log m$
- Empirical complexity $289n^{1.6}m^{0.6}$ reasonably agrees with theoretical worst case.
Statistical tests in a nutshell

- Formulate a null hypothesis and an alternative hypothesis.
- Choose an adequate statistical test.
- Define a significance level.
- Apply the test and accept or reject the null hypothesis.
Example: Binomial test

- Are there born more women than men in a given population?
- For $X = \text{[male born]}$ null hypothesis is $P[X] = 0.5$.
- Take a sample $X_1, \ldots, X_{10}$.
- For independent samples $S = \sum_{i \in [n]} X_i$ distributed binomially

$$B(k; n, p) = \binom{n}{k} p^k (1 - p)^{n-k}$$

- Thus: $S \sim B(k; 10, 0.5)$ if null hypothesis true.
Example: Binomial test (cont.)

<table>
<thead>
<tr>
<th>k</th>
<th>0/10</th>
<th>1/9</th>
<th>2/8</th>
<th>3/7</th>
<th>4/6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P[X = k]$</td>
<td>0.001</td>
<td>0.010</td>
<td>0.044</td>
<td>0.117</td>
<td>0.205</td>
<td>0.246</td>
</tr>
<tr>
<td>$P[X \geq k]$</td>
<td>1.000</td>
<td>0.999</td>
<td>0.989</td>
<td>0.945</td>
<td>0.828</td>
<td>0.623</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>k</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P[X \geq k]$</td>
<td>0.377</td>
<td>0.172</td>
<td>0.055</td>
<td>0.011</td>
<td>0.001</td>
</tr>
</tbody>
</table>

- What is the probability of event $X$, supposing $H_0$ is true?
- Define *significance level* $p$, f.ex. $p = 0.05$.
- *Reject* null hypothesis for events with $p < 0.05$.
- Here: for $X \geq 9$. 
Statistical test 3.1 (Sign test)
Precondition  Two paired samples $x_1, \ldots, x_n$ e $y_1, \ldots, y_n$. Values $x_i - y_i$ are independent and distributed around a common median $m$.

Null hypothesis  $H_0: m = 0$;

Alternative hyp.  $H_1: m > 0$, $m < 0$, $m \neq 0$.

Test statistic  $B = \sum_{i \in [n]} [x_i > y_i]$. 

Observations  Samples $x_i = y_i$ are removed (or half of them attributed to group $x_i > y_i$).
Statistical tests

<table>
<thead>
<tr>
<th>$H_0$ true</th>
<th>$H_0$ maintained</th>
<th>$H_0$ rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>Type 1 error</td>
<td></td>
</tr>
<tr>
<td>Type 2 error</td>
<td>Correct</td>
<td></td>
</tr>
</tbody>
</table>

- Significance level: probability $P[H_0$ rejected $| H_0$ true$]$ of committing type 1 error.
Power of a test

- Power: probability $P[H_0 \text{ rejected} \mid H_1 \text{ true}]$ of detecting $H_1$, i.e. not making type 2 error.
- Depends on magnitude of effect we want to observe.
- Examples: Power to detect $P[X] > q$ above, for $p = 0.05$: \[
\begin{array}{cc}
0.6 & 0.8 \\
0.046 & 0.376
\end{array}
\]
- Power depends on sample size. Example with $n = 50$: \[
\begin{array}{cc}
0.6 & 0.8 \\
0.336 & 0.997
\end{array}
\]
- Sample size large enough to have power of 0.8 considered acceptable.
Choosing a statistical test

- Parametric (known distribution) or non-parametric test.
- Paired or unpaired samples.
  Example paired test: Two deterministic algorithms on same set of instances.
  Example unpaired test: Ten replicates of two randomized algorithms on the same instance.
- Some common tests:

<table>
<thead>
<tr>
<th>Paired</th>
<th>Non-parametric</th>
<th>Sign test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paired</td>
<td>Non-parametric</td>
<td>Wilcoxon signed rank</td>
</tr>
<tr>
<td>Unpaired</td>
<td>Non-parametric</td>
<td>Wilcoxon rank sum (Mann-Whitney U)</td>
</tr>
<tr>
<td>(Un)paired</td>
<td>Parametric</td>
<td>Student’s $t$ test (Gaussian)</td>
</tr>
</tbody>
</table>
Example: Watson et al. (2006)

- Null hypothesis: Tabu search *not better*.
- Alternative hypothesis: Tabu search *is better*.
- 50 instances from Taillard, 10 replications each, Wilcoxon rank sum test.
- Number of instances null hypothesis *could not be rejected*.

<table>
<thead>
<tr>
<th>Group</th>
<th>i-ILS</th>
<th>i-RMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta01-10</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>ta11-20</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ta21-30</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>ta31-40</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>ta41-50</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Statistically: All heuristics competitive with Tabu search.
Contents

1. Introduction

2. Design of heuristics

3. Analysis of heuristics
   - Theoretical analysis
   - Practical analysis
   - Some practical tools
   - Case study
Case study: $P \parallel C_{\text{max}}$

- $n$ tasks of time $t_i$, $m$ machines, minimize makespan

\[
\begin{align*}
\text{min.} & \quad C_{\text{max}} \\
\text{s.t.} & \quad \sum_{j \in [m]} x_{ij} = 1, \quad \forall i \in [n], \\
& \quad \sum_{i \in [n]} x_{ij} t_i \leq C_{\text{max}}, \quad \forall j \in [m], \\
& \quad x_{ij} \in \{0, 1\} \quad \forall i \in [n], j \in [m].
\end{align*}
\]
Example instance
Case study: BCO heuristic

Bee colony optimization for scheduling independent tasks to identical processors

- Maintain $k = 5$ partial solutions ("bees")
- Each: choose unscheduled task with prob. $\alpha t_i$
- Schedule it at processor $j$ with prob. $\alpha C_{\text{max}}^b - C_j^b$
- Repeat $r$ times.
Case study: BCO heuristic

Interaction after $r$ individual steps:

- Normalize makespan 
  \[ N^b = (\overline{C} - C_{\max}^b)/(\overline{C} - \underline{C}) \] 
  with \( \overline{C} = \max_b C_{\max}^b \) and \( \underline{C} = \min_b C_{\max}^b \)

- Abandon current solution in iteration $s$ with prob. \( \exp\left(-\left(\overline{C} - C_{\max}^b\right)/s\right) \).

- ... and choose a non-abandoned one with prob. \( \propto N^b \).
Example: Bee colony optimization

Bees follows randomized trajectory
Example: Bee colony optimization (cont.)

Bee B "recruited" bee 2.
Example: Bee colony optimization (cont.)

Bees continue own randomized trajectory
Scientific approach applied

- **Measure & describe**: Theoretical & empirical complexity? Empirical quality? How good in comparison to LPT?
- **Explain**: How does it work? Why does it work? What are the key components?
- **Innovate**: Relation to other multi-particle heuristics (e.g. beam search, go with the winners)?
Example: Theoretical complexity

- Fixed number of 100 repetitions.
- Sampling from $i$ events: $O(i)$ (naively).
- Copying a solution: $O(i)$ for $i$ tasks.

$$T(n, m, k) = k \sum_{i \in [n]} (O(i) + O(m) + O(i))$$

$$= O(knm + kn^2) = O(kn^2).$$
Example: Runtime of an instance

Instance with $n = 5000$ processors (rand0000).

<table>
<thead>
<tr>
<th>$m$</th>
<th>OPT</th>
<th>Opt CPU time</th>
<th>BCO</th>
<th>BCO % error</th>
<th>BCO min time</th>
<th>BCO CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6844</td>
<td>1.112</td>
<td>6844</td>
<td>0.000</td>
<td>0.070</td>
<td>6.922</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>6.113</td>
<td>3422</td>
<td>0.000</td>
<td>0.209</td>
<td>20.097</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>9.786</td>
<td>1711</td>
<td>0.000</td>
<td>0.217</td>
<td>21.987</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>30.288</td>
<td>1095</td>
<td>0.000</td>
<td>0.226</td>
<td>23.166</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>28.561</td>
<td>548</td>
<td>0.000</td>
<td>0.251</td>
<td>25.056</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>1130.31</td>
<td>277</td>
<td>1.095</td>
<td>0.560</td>
<td>29.585</td>
</tr>
</tbody>
</table>
Example: Runtime, linear model

\[ T(m) = 20.2 + 0.1m \text{ s} \]
Measure & describe: Observations

- All four tested instances show a similar behavior.
- Too few details to determine theoretical complexity.
  - Possible different sampling method (e.g. in $O(\log n)$ using Fenwick trees)
- Article does not explain run-time completely.
- Possible explanations
- Needs *better investigation* to be explained.
Explain: a curious person may ask …

- Dependency on the number of bees?
- Larger or smaller forward steps?
- What does the interaction of bees contribute?
  Could simple repetitions be similarly efficient?
- Heuristic performance of ILP solver?
- Dependency on the LPT rule?
- Article answers *none of the above*. 
Some tentative answers

- Straightforward re-implementation.

- $b = 1, 2, 4, 5, 10, 20, 40$ bees and $br = 500$ held constant.

- 11 replications for case $m = 100$. 
Dependence on number of bees
No qualitatively or quantitatively relevant difference in both cases.

Seems to indicate: simple randomized LPT allocation *performs equally well*.

Observe: case $b = 1$ has no interaction.

Seems to indicate: *interaction may be ineffective*.

Consistent with observation that *step size seems insignificant*.

Caution: Needs some more investigation.
Innovate

- Model very similar to heuristics in *tree models*.
- Search variant: *Go with the winners* (Aldous and Vazirani 1994).
Theorem 3.1 (Aldous and Vazirani (1994))
For a tree of depth \( D \), with vertices \( V_i \) at depth \( i \), let \( p(v) \) be the probability visiting vertex \( v \) in a random walk from the root to the leaves for a given probability distribution \( p(u \mid v) \) over the children \( u \) of each internal vertex \( v \). Let \( \kappa = \max_{0 \leq i < j \leq D} \kappa_{i,j} \) com

\[
\kappa_{i,j} = \frac{P[d \geq i]}{P[d \geq j]^2} \sum_{v \in V_i} p(v) P[d \geq j \mid v]^2.
\]

Then GWTW with \( B = \kappa D^{O(1)} \) particles fails to reach depth \( D \) with probability at most \( 1/4 \).
By Aldous and Vazirani (1994): GWTW can exhibit *exponential speedup* over single-point search.

*Is it possible to demonstrate a similar theorem for BCO?*
Solution times & quality: a final look

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>CPLEX $t$ (s)</th>
<th>val.</th>
<th>dev.</th>
<th>$t_{\text{min}}$</th>
<th>BCO $\bar{t}$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6844</td>
<td>1.112</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
<td>6922</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>6.113</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
<td>20097</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>9.786</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
<td>21987</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>30.288</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
<td>23166</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>28.561</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
<td>25056</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>1130.31</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
<td>29585</td>
</tr>
</tbody>
</table>

- Small relative deviation from optimum.
- Time 70–30000 ms.
- Much better than CPLEX.
## Solution times & quality: a final look

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>CPLEX $t$ (s)</th>
<th>BCO val.</th>
<th>BCO dev.</th>
<th>$t_{\text{min}}$</th>
<th>$\bar{t}$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6844</td>
<td>1.112</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
<td>6922</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>6.113</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
<td>20097</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>9.786</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
<td>21987</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>30.288</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
<td>23166</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>28.561</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
<td>25056</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>1130.31</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
<td>29585</td>
</tr>
</tbody>
</table>

- Small relative deviation from optimum.
- Time 70–30000 ms.
- **Much better** than CPLEX.
- **Really?**
What about using **CPLEX as a heuristic**?

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO</th>
<th>CPLEX, 1thr., 60s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>val.</td>
<td>dev.</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
</tr>
</tbody>
</table>

- Starts being competitive.
What about using *CPLEX as a heuristic*?

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO val.</th>
<th>BCO dev.</th>
<th>$t_{\min}$ (s)</th>
<th>$t$ (ms)</th>
<th>CPLEX, 1thr., 60s val.</th>
<th>CPLEX, 1thr., 60s dev.</th>
<th>$t$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
<td>6922</td>
<td>6844</td>
<td>0.000</td>
<td>850</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
<td>20097</td>
<td>3422</td>
<td>0.000</td>
<td>3060</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
<td>21987</td>
<td>1711</td>
<td>0.000</td>
<td>7640</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
<td>23166</td>
<td>1096</td>
<td>0.009</td>
<td>60000</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
<td>25056</td>
<td>549</td>
<td>0.183</td>
<td>60000</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
<td>29585</td>
<td>281</td>
<td>2.555</td>
<td>60000</td>
</tr>
</tbody>
</table>

- Starts being competitive.
What about using a *more compact model*?

Let’s work with item multiplicities.

<table>
<thead>
<tr>
<th>m</th>
<th>Opt</th>
<th>BCO</th>
<th>CPLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>val.</td>
<td>dev.</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
</tr>
</tbody>
</table>

- Done: exact and 10 times faster.
What about using a *more compact model*? Let’s work with item multiplicities.

<table>
<thead>
<tr>
<th>( m )</th>
<th>Opt</th>
<th>BCO val.</th>
<th>BCO dev.</th>
<th>( t_{\min} )</th>
<th>CPLEX val.</th>
<th>BCO t</th>
<th>CPLEX dev.</th>
<th>CPLEX t</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
<td>6922</td>
<td>6844</td>
<td>0.000</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
<td>20097</td>
<td>3422</td>
<td>0.000</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
<td>21987</td>
<td>1711</td>
<td>0.000</td>
<td>30</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
<td>23166</td>
<td>1095</td>
<td>0.000</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
<td>25056</td>
<td>548</td>
<td>0.000</td>
<td>80</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
<td>29585</td>
<td>277</td>
<td>0.000</td>
<td>60</td>
</tr>
</tbody>
</table>

Done: exact and 10 times faster.
A recent paper (Davidović et al. 2012) presented a bee colony metaheuristic for scheduling independent tasks to identical processors, evaluating its performance on a benchmark set of instances from the literature. We examine two exact algorithms from the literature, the former published in 1995, the latter in 2008 (and not cited by the authors). We show that both such algorithms solve to proven optimality all the considered instances in a computing time that is several orders of magnitude smaller than the time taken by the new algorithm to produce an approximate solution.
<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO</th>
<th>DM</th>
<th>List/LPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val.</td>
<td>dev.</td>
<td>$t_{\text{min}}$</td>
<td>$t$</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
</tr>
</tbody>
</table>
### Solution times & quality (cont.)

Dell’Amico et al. (2012): exact and 100 times faster

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO</th>
<th>DM</th>
<th>List/LPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>val.</td>
<td>dev.</td>
<td>$t_{\text{min}}$</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
</tr>
</tbody>
</table>
### Solution times & quality (cont.)

Even list scheduling & LPT: exact and > 10 times faster

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO</th>
<th>DM</th>
<th>List/LPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>val.</td>
<td>dev.</td>
<td>$t_{\text{min}}$</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
</tr>
</tbody>
</table>
### Solution times & quality (cont.)

<table>
<thead>
<tr>
<th>$m$</th>
<th>Opt</th>
<th>BCO</th>
<th>DM</th>
<th>List/LPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val.</td>
<td>dev.</td>
<td>$t_{min}$</td>
<td>$t$</td>
</tr>
<tr>
<td>4</td>
<td>6844</td>
<td>6844</td>
<td>0.000</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>3422</td>
<td>3422</td>
<td>0.000</td>
<td>209</td>
</tr>
<tr>
<td>16</td>
<td>1711</td>
<td>1711</td>
<td>0.000</td>
<td>217</td>
</tr>
<tr>
<td>25</td>
<td>1095</td>
<td>1095</td>
<td>0.000</td>
<td>226</td>
</tr>
<tr>
<td>50</td>
<td>548</td>
<td>548</td>
<td>0.000</td>
<td>251</td>
</tr>
<tr>
<td>100</td>
<td>274</td>
<td>277</td>
<td>1.095</td>
<td>560</td>
</tr>
</tbody>
</table>

By the way: list scheduling is 2-, LPT 4/3-approximation (Graham 1966).
Wir stehen selbst enttäuscht und sehnen
betroffen
Den Vorhang zu und alle Fragen offen.

Ficamos triste também ao notar,
por nosso lado
Tanto problema em aberto
e a cortina fechada.

Brecht – A alma boa de Setsuan
Some pointers: Problem solving

How to Solve It: A New Aspect of Mathematical Method
By G. Polya

How to Solve It: Modern Heuristics
Second Edition
By Zbigniew Michalewicz and David B. Fogel
Statistical Analysis of Computational Tests of Algorithms and Heuristics

Marie Coffin and Matthew J. Saltzman / Department of Mathematical Sciences, Clemson University, Clemson, SC 29634-0975, Email: mcoffin@clemson.edu, mjs@clemson.edu

(Received: July 1998; revised: November 1998, March 1999; accepted: November 1999)
Conclusions

- Be critical and curious.
- Compare & evaluate: don’t be the “bee colony” guy, rather be a “mythbuster”.
- Present *qualitative* results and *quantitative* statistical evaluation (not too hard: use it!).
- Search for *explanations* with *reach*: “why” matters more than “that”.
There ain’t no such thing as a free lunch.

Thanks for your attention!
Aldous, David and Umesh Vazirani (1994). ““Go With the Winners” Algorithms”. In: *Proc. 26th STOC*.


