A Generic Bet-and-run Strategy for Speeding Up Stochastic Local Search

Markus Wagner  
markus.wagner@adelaide.edu.au

Code and results:  https://bitbucket.org/markuswagner/restarts
Context in this session

Carola: change parameters during a run
Anja: change algorithms during a run
Markus: don’t change anything during a run

Speeding Up Stochastic Local Search

Markus Wagner
markus.wagner@adelaide.edu.au

Code and results: https://bitbucket.org/markuswagner/restarts
Restarts

A desktop PC does not work properly → we restart it.

Performance of stochastic algorithm and randomized search heuristics unsatisfactory → we restart it again and again.

While this approach is well-known, few algorithms directly incorporate such restart strategies.

Potential reason: added complexity of designing an appropriate restart strategy that is advantageous for the considered algorithm.

We are looking for: a generic framework for restart strategies that is not overly dependent on the algorithm used and the problem considered.
Related work

Luby, Sinclair, and Zuckerman (1993)
- for Las Vegas algorithms with known run time distribution:
  sequence of running times $(1,1,2,1,1,2,4,1,1,2,1,1,2,4,8,...)$ optimal restarting strategy (up to constant factors)

Satisfiability problem
- empirical comparisons showing substantial impact on efficiency of SAT solvers [Biere 2008, Huang 2007]
- unsurprising as SAT/CSP solvers learn no-goods during backtracking [Ciréé et al 2014]

Classic optimisation algorithms are often deterministic
- The underlying algorithm of IBM ILOG CPLEX is not random, but characteristics change with memory constraints and parallel computations.
- Lalla-Ruiz and Voss (2016) investigated different mathematical programming formulations to provide different starting points.
Related work
Bet-and-Run by Fischetti and Monaci (2014)

Phase 1 of length $k \cdot t_1$
Phase 2 of length $t_2 = t - k \cdot t_1$

$k$ runs
$t_1$
$t_1 + t_2$
time

Notes
Single-run:
$k = 1$
Multi-run with restarts from scratch:
$t_1 = t / k$ and $t_2 = 0$

Fischetti and Monaci (2014)
“Exploiting erraticism in search”
k = 5, CPLEX, diversity, MIPlib 2010

de Perthuis de Laillevault, Doerr, and Doerr (2015)
1+1-EA on OneMax
possible additive runtime gain of order $\sqrt{n \log n}$
Related work
Bet-and-Run by Fischetti and Monaci (2014)

Phase 1 of length $k \cdot t_1$
Phase 2 of length $t_2 = t - k \cdot t_1$

$k$ runs

Notes
Single-run:
- $k=1$

Multi-run with restarts from scratch:
- $t_1 = t/k$ and $t_2 = 0$

Fischetti and Monaci (2014)
"Exploiting erraticism in search"
- $k=5$, CPLEX, diversity, MIPLib 2010

De Perthuis de Laillevault, Doerr, and Doerr (2015)
1+1 EA on OneMax
- possible additive runtime gain of order $\sqrt{n \log n}$
Related work
Bet-and-Run by Fischetti and Monaci (2014)

Implementation Detail:
The initial runs can be run sequentially – they don’t have to be in parallel. Keep in mind: our goal is to make best use of some total computation budget $t$, not of some wallclock time.

Single-run:
$k=1$

Multi-run with restarts from scratch:
$t_1 = t/k$ and $t_2 = 0$

Fischetti and Monaci (2014)
“Exploiting erraticism in search”
k=5, CPLEX, diversity, MIPlib 2010

de Perthuis de Laillevault, Doerr, and Doerr (2015)
1+1 EA on OneMax
possible additive runtime gain of order $\sqrt{n \log n}$
A Generic Bet-and-Run Strategy

Our experiments

- **Traveling Salesperson**
  - Lin-Kernighan Heuristic (from Concorde)
  - 111 symmetric TSPlib instances with up to 100k cities
- **Minimum Vertex Cover**
  - FastVC (Cai 2015)
  - 86 large MVC instances

- Also, algorithms are **pure black boxes**: start with seed … … … … … stop

- Lots of bet-and-run strategies
  - Example: heatmap on the right
  - \(~450\) bet-and-run setups for 1 instance

Example: FastVC on MVC instance shipsec1.mtx
Total budget t=240s
Shown in colour is absolute distance to best-found (117,366).
A Generic Bet-and-Run Strategy
Dependency on total time budget

Example: solution quality achieved by FastVC on instance sc-shipsec5 (average of 100 runs)
Cross Domain Study
To-be-investigated Bet-And-Run Approaches

$\text{RESTARTS}_x^k$ refers to the strategy where $k$ initial runs are performed, and each of the runs has a computational budget of $x\%$ of the total time budget.

$\text{RESTARTS}_{\text{Luby}}^k$ refers to the strategy that uses in its first phase runs whose lengths are defined by the Luby sequence. $k$ refers to the sequence length used in the first phase, and each Luby time unit is $x\%$ of the total time.
Cross Domain Study
First Results (10 instances per domain only, 14 strategies)

<table>
<thead>
<tr>
<th>Budget: $400 \cdot t_{\text{init}}$</th>
<th>TSP</th>
<th>MVC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESTARTS</strong>$^1_{100%}$</td>
<td>12.3</td>
<td>9.4</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^4_{25%}$</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^4_{10%}$</td>
<td>3.7</td>
<td>4.4</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^{10}_{4%}$</td>
<td>2.3</td>
<td>1.8</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^{40}_{1%}$</td>
<td>3.0</td>
<td>1.3</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^4_{2.5%}$</td>
<td>6.2</td>
<td>5.2</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^{10}_{1%}$</td>
<td>5.6</td>
<td>3.2</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^{40}_{0.25%}$</td>
<td>7.7</td>
<td>3.7</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^4_{1%}$</td>
<td>9.9</td>
<td>6.5</td>
</tr>
<tr>
<td><strong>RESTARTS</strong>$^{10}_{0.4%}$</td>
<td>9.0</td>
<td>4.5</td>
</tr>
<tr>
<td><strong>RESTARTS Luby</strong>$^4_{1%}$</td>
<td>10.8</td>
<td>6.4</td>
</tr>
<tr>
<td><strong>RESTARTS Luby</strong>$^{10}_{1%}$</td>
<td>10.3</td>
<td>3.5</td>
</tr>
<tr>
<td><strong>RESTARTS Luby</strong>$^{40}_{1%}$</td>
<td>7.2</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Shown are average ranks across 10 instances.
More tables in the paper.
Cross Domain Study
Summary (~200 instances, 1 Bet-and-Run strategy vs 1 single run)

Universally good (given our experiments): Restarts^{40\%}
\begin{itemize}
    \item Phase 1: 40 runs, each with a time budget of 1\% of the total time budget
    \item Phase 2: use the remaining 60\% to continue the best run of Phase 1
\end{itemize}

Comparison of our “universal” Restarts^{40\%} with a single run:
Wilcoxon-rank-sum test (p=0.05): green shows where Restarts^{40\%} is significantly better,
grey (identical or insignificant), red (single run is better)

Explicable erraticism using restarts:

- TSP:
  - 69 successes, 35 failures
  - 73 successes, 34 failures
  - 74 successes, 33 failures

- MVC:
  - 31 successes, 6 failures
  - 35 successes, 6 failures
  - 36 successes, 5 failures

Total time limit:
- 100 \cdot t_{init}
- 400 \cdot t_{init}
- 1000 \cdot t_{init}
Summary so far

We studied a generic bet-and-run restart strategy

- easy to implement as an additional speed-up heuristic
- demonstrated effectiveness on two classical NP-complete optimisation problems with state-of-the-art solvers
- Significant advantage of **Restarts**$^{40 \times 1\%}$:
  - Phase 1: 40 runs with 1% each of the total time
  - Phase 2: continue the best of these 40 for 60% of the total time

Published:

AAAI Conference on Artificial Intelligence 2017
A Generic Bet-and-run Strategy for Speeding Up Stochastic Local Search
*Tobias Friedrich, Timo Kötzing, and Markus Wagner*

Code and results: [https://bitbucket.org/markuswagner/restarts](https://bitbucket.org/markuswagner/restarts)
More work on this (1/3) – Theory

Genetic and Evolutionary Computation Conference (GECCO) 2017
Theoretical results on bet-and-run as an initialisation strategy
Andrei Lissovoi, Dirk Sudholt, Markus Wagner, and Christine Zarges

We define a family of pseudo-Boolean functions ($\phi$):
- the plateau shows a high fitness, but does not allow for further progression
- the slope has a low fitness initially, but does lead to the global optimum.

Results:
- non-trivial $k$ and $t_1$ are necessary,
- $t_1$ is linked to properties of the function,
- fixed budget analysis to guide selection of the bet-and-run parameters to maximise expected fitness after $t = k \cdot t_1 + t_2$ fitness evaluations.

$$f_h(x) = \begin{cases} 
|x|_1 & \text{if } |x|_1 > n/2 \\
h & \text{otherwise}
\end{cases}$$
More work on this (2/3) – **Generalised Bet-and-Run**

**AAA 2019**
An Improved Generic Bet-and-Run Strategy for Speeding Up Stochastic Local Search
Thomas Weise, Zijun Wu, and Markus Wagner

Major result of 78 million experiments:
Decision maker “take current best” is difficult to beat, but it is possible.
Drawback of previous work: Whether a run looks promising or abysmal, it gets run exactly until the predetermined limit is reached.

We train (offline) a controller. It then decides online:
1. Continue the current run.
2. Continue an old run.
→ It considers: performance and performance projections of the individual runs, and the remaining time budget.
More work on this (4/3) – Future work

- Other domains: continuous optimisation, multi-objective optimisation, ...
- Heterogeneous setups:
  - different hierarchies/races/... of the independent runs
  - different algorithms
  - different algorithm configurations
  - configure on-the-fly
    ➔ this might be a hot topic, and it has a connection to algorithm control, hyper-heuristics, partial restarts (perturbations), ...

Markus Wagner
markus.wagner@adelaide.edu.au

Code and results: https://bitbucket.org/markuswagner/restarts