Some "Do"s and "Don't"s of Benchmarking

Paul Shaw, Program Manager, CP product development, IBM.
paul.shaw@fr.ibm.com

WebSphere software

Optimization at IBM

- IBM Research has a tradition of optimization
  - Probably most recently for COIN-OR
- ILOG was fully transferred to IBM just under one year ago
  - Brought new optimization products to IBM
  - Since 4th of June, IBM sells “CPLEX Optimization Studio”
    - Comprises CPLEX, CP Optimizer, OPL
  - As well as ILOG CP (the older CP products Solver, Scheduler, Dispatcher)
- Academic Initiative
  - Full CPLEX Optimization Studio will be free for academics
    - https://www.ibm.com/developerworks/university/academicinitiative
    - https://www.ibm.com/developerworks/university/support/faqs.html

CP Optimizer

- CP Optimizer is a constraint programming engine concentrating on
  - Combinatorial optimization problems
  - Scheduling problems

CP Optimizer

- CP Optimizer is a constraint programming engine concentrating on
  - Combinatorial optimization problems
  - Scheduling problems
- CP Optimizer has a robust built-in search engine (sometimes referred to as *autonomous search*)
  - Although the search can be fully programmed if desired
  - Concise hints on search can also be given

CP Optimizer

- CP Optimizer is a constraint programming engine concentrating on
  - Combinatorial optimization problems
  - Scheduling problems
- CP Optimizer has a robust built-in search engine (sometimes referred to as *autonomous search*)
  - Although the search can be fully programmed if desired
  - Concise hints on search can also be given
- Our team concentrates on:
  - Making CP Optimizer solve more quickly
  - Making CP Optimizer easier to use
  - Adding new modelling or solving features

About this talk, or, “sorry for stealing the idea”

- Fifteen years ago
  - I worked in a research group called APES
    - Algorithms, Problems, Empirical Studies
  - We studied algorithms and did a lot of experiments (or if you like, *benchmarking*)
  - One report we wrote was called “How Not To Do It”
    - Informally chronicled some misadventures in the world of experiments on NP-hard problems

About this talk, or, “sorry for stealing the idea”

- Fifteen years ago
  - I worked in a research group called APES
    - Algorithms, Problems, Empirical Studies
  - We studied algorithms and did a lot of experiments (or if you like, *benchmarking*)
  - One report we wrote was called “How Not To Do It”
    - Informally chronicled some misadventures in the world of experiments on NP-hard problems
- Today
  - I tried (without peeking at the report) to remember some of the themes and to see how they applied to me now
  - A couple of themes are new

Scenario

- You are are PhD student working on a research area proposed to you by your thesis advisor. You've thought of a cool new algorithm for a well-known problem class. Eager to see how it performs, you code it up and run a load of experiments on classic benchmarks over the weekend.
- You check on the results on Monday morning. Tremendous! You close several open problems by proving the optimality of some known upper bounds.
- Do you:

Scenario

- You are are PhD student working on a research area proposed to you by your thesis advisor. You've thought of a cool new algorithm for a well-known problem class. Eager to see how it performs, you code it up and run a load of experiments on classic benchmarks over the weekend.
- You check on the results on Monday morning. Tremendous! You close several open problems by proving the optimality of some known upper bounds.
- Do you:
  - A) Call your thesis advisor to thank them for being so insightful in proposing the area. He or she surely deserves at least half the credit.

Scenario

- You are are PhD student working on a research area proposed to you by your thesis advisor. You've thought of a cool new algorithm for a well-known problem class. Eager to see how it performs, you code it up and run a load of experiments on classic benchmarks over the weekend.
- You check on the results on Monday morning. Tremendous! You close several open problems by proving the optimality of some known upper bounds.
- Do you:
  - A) Call your thesis advisor to thank them for being so insightful in proposing the area. He or she surely deserves at least half the credit.
  - B) Allow yourself a little smile – you always knew your idea was brilliant.

Scenario

- You are are PhD student working on a research area proposed to you by your thesis advisor. You've thought of a cool new algorithm for a well-known problem class. Eager to see how it performs, you code it up and run a load of experiments on classic benchmarks over the weekend.
- You check on the results on Monday morning. Tremendous! You close several open problems by proving the optimality of some known upper bounds.
- Do you:
  - A) Call your thesis advisor to thank them for being so insightful in proposing the area. He or she surely deserves at least half the credit.
  - B) Allow yourself a little smile – you always knew your idea was brilliant.
  - C) Call up your friends to go out and celebrate – the thesis is in the bag.

Scenario

- You are are PhD student working on a research area proposed to you by your thesis advisor. You've thought of a cool new algorithm for a well-known problem class. Eager to see how it performs, you code it up and run a load of experiments on classic benchmarks over the weekend.
- You check on the results on Monday morning. Tremendous! You close several open problems by proving the optimality of some known upper bounds.
- Do you:
  - A) Call your thesis advisor to thank them for being so insightful in proposing the area. He or she surely deserves at least half the credit.
  - B) Allow yourself a little smile – you always knew your idea was brilliant.
  - C) Call up your friends to go out and celebrate – the thesis is in the bag.
  - D) Start scanning your code for bugs.

DON'T TRUST YOURSELF

DON'T TRUST YOURSELF

- DO have healthy skepticism
  - DON'T believe that someone else will volunteer to find errors in your work

DON'T TRUST YOURSELF

- DO have healthy skepticism
  - DON'T believe that someone else will volunteer to find errors in your work
- DO check everything twice, then check it again (including the problem spec.)

DON'T TRUST YOURSELF

- DO have healthy skepticism
  - DON'T believe that someone else will volunteer to find errors in your work
- DO check everything twice, then check it again (including the problem spec.)
- DON'T do it alone
  - DO get help in your group to check your logic, code and algorithm
  - DO use tools to detect errors, but DO write simple code

DON'T TRUST YOURSELF

- DO have healthy skepticism
  - DON'T believe that someone else will volunteer to find errors in your work
- DO check everything twice, then check it again (including the problem spec.)
- DON'T do it alone
  - DO get help in your group to check your logic, code and algorithm
  - DO use tools to detect errors, but DO write simple code
- DO write a solution checker where appropriate (or better, use someone else's)
  - DO resist the temptation to use the same data and solution reader

DON'T TRUST YOURSELF

- DO have healthy skepticism
  - DON'T believe that someone else will volunteer to find errors in your work
- DO check everything twice, then check it again (including the problem spec.)
- DON'T do it alone
  - DO get help in your group to check your logic, code and algorithm
  - DO use tools to detect errors, but DO write simple code
- DO write a solution checker where appropriate (or better, use someone else's)
  - DO resist the temptation to use the same data and solution reader
- DO write a second implementation
- DO construct a proof
- ...

DON'T TRUST YOURSELF: A study

- Solve the optimization version of the MAX-CUT problem on a cubic graph
  - Best known specialized algorithm has complexity $O^*(2^{m/6})$
- We wanted to try CP Optimizer to see how it compared empirically

DON'T TRUST YOURSELF: A study

- Solve the optimization version of the MAX-CUT problem on a cubic graph
  - Best known specialized algorithm has complexity $O^*(2^{m/6})$
- We wanted to try CP Optimizer to see how it compared empirically
- Results:
  - *Obvious model* was terrible
    - from memory, growth was around $2^{m/2}$

DON'T TRUST YOURSELF: A study

- Solve the optimization version of the MAX-CUT problem on a cubic graph
  - Best known specialized algorithm has complexity $O^*(2^{m/6})$
- We wanted to try CP Optimizer to see how it compared empirically
- Results:
  - *Obvious model* was terrible
    - from memory, growth was around $2^{m/2}$
  - Second model included a dominance rule to cut non-optimal solutions
    - better: growth around $2^{m/5}$

DON'T TRUST YOURSELF: A study

- Solve the optimization version of the MAX-CUT problem on a cubic graph
  - Best known specialized algorithm has complexity $O^*(2^{m/6})$
- We wanted to try CP Optimizer to see how it compared empirically
- Results:
  - *Obvious model* was terrible
    - from memory, growth was around $2^{m/2}$
  - Second model included a dominance rule to cut non-optimal solutions
    - better: growth around $2^{m/5}$
  - Third model included a more sophisticated dominance rule
    - much faster: growth was around m^3

DON'T TRUST YOURSELF: A study

- Solve the optimization version of the MAX-CUT problem on a cubic graph
  - Best known specialized algorithm has complexity $O^*(2^{m/6})$
- We wanted to try CP Optimizer to see how it compared empirically
- Results:
  - *Obvious model* was terrible
    - from memory, growth was around $2^{m/2}$
  - Second model included a dominance rule to cut non-optimal solutions
    - better: growth around $2^{m/5}$
  - Third model included a more sophisticated dominance rule
    - much faster: growth was around m^3
- So, I started looking for bugs in the model

DON'T TRUST YOURSELF: How we debugged that one

- Three facts
  - I suspected the third model was pruning too many branches
  - I had two other simpler models
  - I had a method for generating a nearly unlimited number of problems

DON'T TRUST YOURSELF: How we debugged that one

- Three facts
  - I suspected the third model was pruning too many branches
  - I had two other simpler models
  - I had a method for generating a nearly unlimited number of problems
- Generate large numbers of small problems until the more sophisticated algorithm produces a different answer from the simple one
  - Keep the problems as small as possible as some detailed analysis is needed afterwards

DON'T TRUST YOURSELF: How we debugged that one

- Three facts
  - I suspected the third model was pruning too many branches
  - I had two other simpler models
  - I had a method for generating a nearly unlimited number of problems
- Generate large numbers of small problems until the more sophisticated algorithm produces a different answer from the simple one
  - Keep the problems as small as possible as some detailed analysis is needed afterwards
- DO have a *more trusted* implementation
  - Use it as a sanity check

DON'T TRUST YOURSELF: How we debugged that one

- Three facts
  - I suspected the third model was pruning too many branches
  - I had two other simpler models
  - I had a method for generating a nearly unlimited number of problems
- Generate large numbers of small problems until the more sophisticated algorithm produces a different answer from the simple one
  - Keep the problems as small as possible as some detailed analysis is needed afterwards
- DO have a *more trusted* implementation
  - Use it as a sanity check
- DO look for counter examples (automatically, or by hand)

DON'T TRUST YOURSELF: How we debugged that one

- Three facts
  - I suspected the third model was pruning too many branches
  - I had two other simpler models
  - I had a method for generating a nearly unlimited number of problems
- Generate large numbers of small problems until the more sophisticated algorithm produces a different answer from the simple one
  - Keep the problems as small as possible as some detailed analysis is needed afterwards
- DO have a *more trusted* implementation
  - Use it as a sanity check
- DO look for counter examples (automatically, or by hand)
- DO test as widely as possible

Progressive Party Problem

- Organize a party in a marina on a number of *host boats*
  - Each boat has a *capacity* (people) and a crew of a certain size
  - The party is organized into six (or more periods)
  - Host crews stay on their host boat – each guest crew visits a new host boat at each period

Progressive Party Problem

- Organize a party in a marina on a number of *host boats*
  - Each boat has a *capacity* (people) and a crew of a certain size
  - The party is organized into six (or more periods)
  - Host crews stay on their host boat – each guest crew visits a new host boat at each period
- Constraints
  - The total size of host and guest crews on a boat is less than boat capacity
  - Each guest crew must visit a different boat in each period
  - No two guest crews can meet more than once

Progressive Party Problem

- Organize a party in a marina on a number of *host boats*
  - Each boat has a *capacity* (people) and a crew of a certain size
  - The party is organized into six (or more periods)
  - Host crews stay on their host boat – each guest crew visits a new host boat at each period
- Constraints
  - The total size of host and guest crews on a boat is less than boat capacity
  - Each guest crew must visit a different boat in each period
  - No two guest crews can meet more than once
- Objective: minimize the number of *host boats*
  - Decide on the host boats and a visit schedule for the guest crews

Progressive Party Problem

- The progressive part problems can be considered to have two aspects:
  - (a) Decide on the set of host boats
  - (b) Given the host boats, decide on a schedule for the guest crews

Progressive Party Problem

- The progressive part problems can be considered to have two aspects:
  - (a) Decide on the set of host boats
  - (b) Given the host boats, decide on a schedule for the guest crews
- To simplify the problem, solution techniques typically divide the two problems, with normally (a) being solved by hand (*e.g.* using the biggest boats)

Progressive Party Problem

- The progressive part problems can be considered to have two aspects:
  - (a) Decide on the set of host boats
  - (b) Given the host boats, decide on a schedule for the guest crews
- To simplify the problem, solution techniques typically divide the two problems, with normally (a) being solved by hand (*e.g.* using the biggest boats)
- I was pretty ignorant of the literature and just coded up the whole model and used CP Optimizer's search

Progressive Party Problem

```
! ----------------------------------------------------------------------------
! Minimization problem - 1408 variables, 15805 constraints
! Preprocessing : 42 extractables eliminated, 42 constraints generated
! LogPeriod             = 10000
! Initial process time : 0.10s (0.08s extraction + 0.02s propagation)
!  . Log search space  : 4408.7 (before), 4235.2 (after)
!  . Memory usage      : 4.9 Mb (before), 7.2 Mb (after)
!  . Variables fixed   : 42
! ----------------------------------------------------------------------------
!   Branches  Non-fixed               Branch decision                   Best
       10000         37                M13,30  =    3
*      12605      1.57s                 M0,20  =    3                     21
*      18049      2.36s                 M0,20  =    0                     20
       20000        321                   H13  =    0                     20
*      20767      2.72s                M21,27  =    1                     19
*      21494      2.80s                M21,27  =    1                     18
*      22756      2.99s                M13,41  =    0                     17
*      27350      3.47s                M23,38  =    1                     16
       30000        385                 T0,31  =    7                     16
*      34262      4.56s                M21,27  =    1                     15
       40000        411                 T3,30  =   32   F                 15
       50000        399                M16,38  =    5                     15
*      50492      7.43s                 M0,15  =    1                     14
       60000        409                  M1,8 !=    1                     14
       70000        462                M11,36  =    5                     14
*      75638     11.54s                 M0,15  =    1                     13
! Search terminated, replaying optimal solution
! ----------------------------------------------------------------------------
! Solution status        : Terminated normally, optimal found (tol. = 0)
! Number of branches     : 75638
! Number of fails        : 17715
! Total memory usage     : 11.9 Mb (10.3 Mb CP Optimizer + 1.6 Mb Concert)
! Time spent in solve    : 11.55s (11.47s engine + 0.08s extraction)
! Search speed (br. / s) : 6594.4
! ----------------------------------------------------------------------------
```

Progressive Party Problem: identify decision variables

```
! ----------------------------------------------------------------------------
! Minimization problem - 1408 variables, 15805 constraints, 1 phase
! Preprocessing : 42 extractables eliminated, 42 constraints generated
! LogPeriod               = 10000
! Initial process time : 0.13s (0.11s extraction + 0.02s propagation)
!  . Log search space  : 4408.7 (before), 4235.2 (after)
!  . Memory usage      : 4.9 Mb (before), 7.2 Mb (after)
!  . Variables fixed   : 42
! ----------------------------------------------------------------------------
!   Branches  Non-fixed                Branch decision                    Best
*       2887      0.48s                  M4,19  =    5                      20
*       4578      0.66s                   M6,7  =    1                      16
*       6625      0.91s                 M22,23  =    0                      14
       10000        714                  T3,28  =   13  F                   14
*      11592      1.53s                  M6,13  =    2                      13
! Search terminated, replaying optimal solution
! ----------------------------------------------------------------------------
! Solution status        : Terminated normally, optimal found (tol. = 0)
! Number of branches     : 11592
! Number of fails        : 3227
! Total memory usage     : 11.1 Mb (9.5 Mb CP Optimizer + 1.6 Mb Concert)
! Time spent in solve    : 1.54s (1.43s engine + 0.11s extraction)
! Search speed (br. / s) : 8106.3
! ----------------------------------------------------------------------------
```

DON'T FORGET TO TRY THE OBVIOUS

DON'T FORGET TO TRY THE OBVIOUS

- It doesn't take long
  - So, even if it works poorly, you did not waste too much time
  - Gives you a simple “trusted” implementation that you can test against

DON'T FORGET TO TRY THE OBVIOUS

- It doesn't take long
  - So, even if it works poorly, you did not waste too much time
  - Gives you a simple “trusted” implementation that you can test against
- It might work well. If the obvious approach has not worked for others:
  - The reasons it did not work might not exist today
  - For the PPP, CP Optimizer is using
    - A global packing constraint
    - A search process which uses restarts and learning
  - Which were not available / used in original studies

DON'T FORGET TO TRY THE OBVIOUS

- It doesn't take long
  - So, even if it works poorly, you did not waste too much time
  - Gives you a simple “trusted” implementation that you can test against
- It might work well. If the obvious approach has not worked for others:
  - The reasons it did not work might not exist today
  - For the PPP, CP Optimizer is using
    - A global packing constraint
    - A search process which uses restarts and learning
  - Which were not available / used in original studies
- In any case, if the obvious approach is a success
  - DON'T TRUST YOURSELF – check your work!

Magic Squares

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

Magic Squares

7	18	25	4	11	65
5	8	19	12	21	65
16	24	13	9	3	65
22	14	2	17	10	65
15	1	6	23	20	65

Magic Squares

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20
65	65	65	65	65

Magic Squares

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

65 65

Diagonally Ordered Magic Squares

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

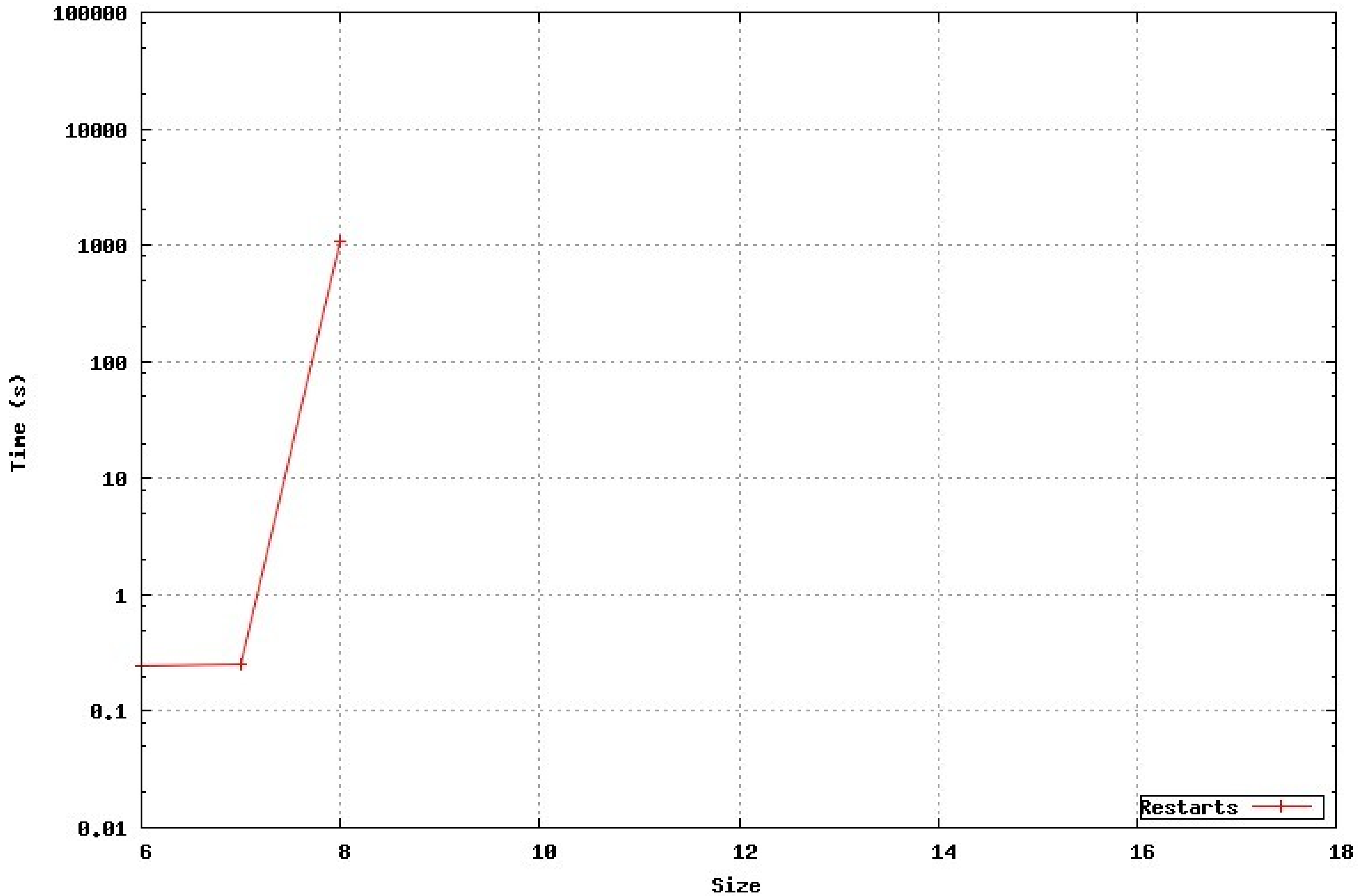
Diagonally Ordered Magic Squares

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

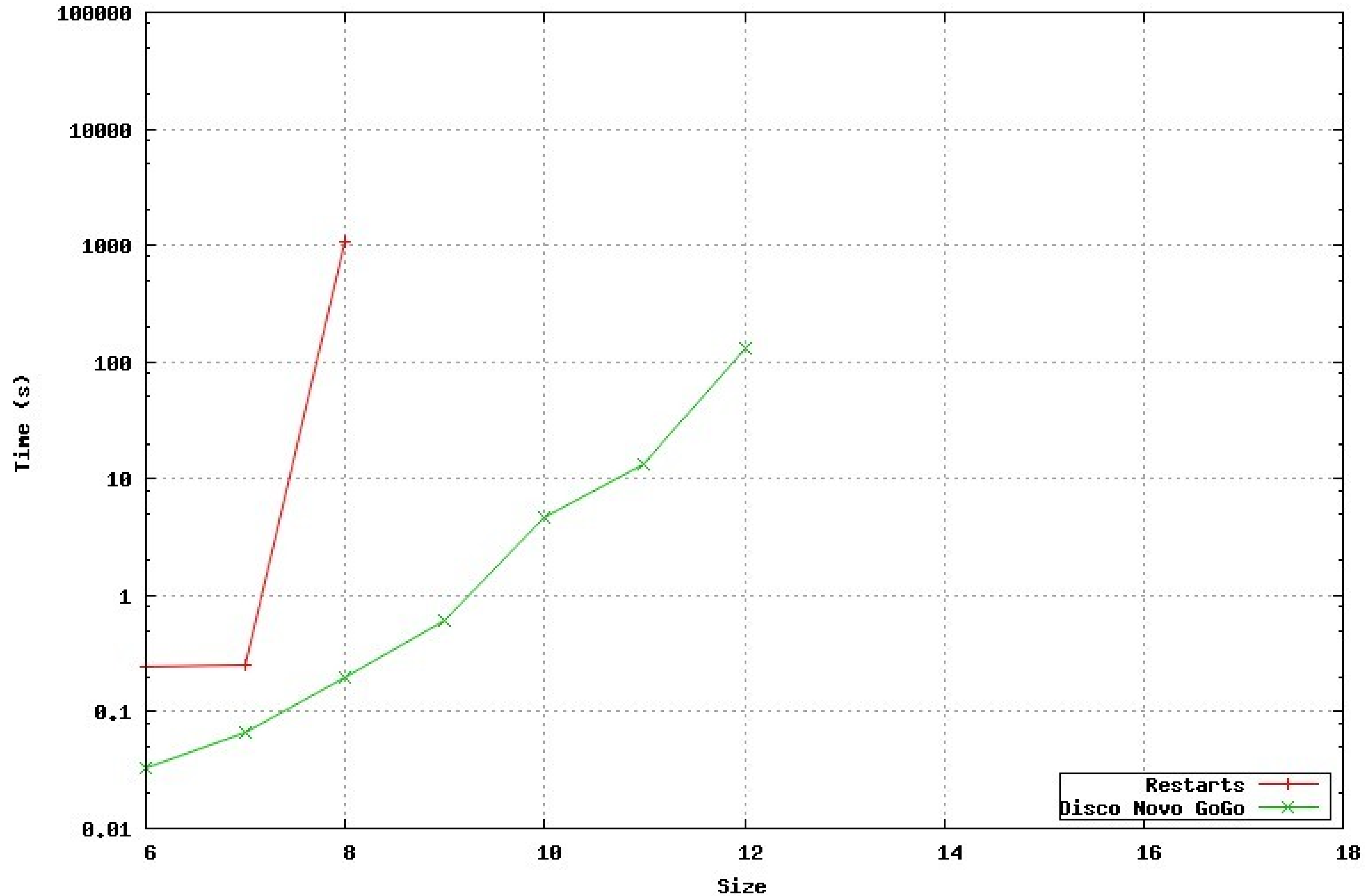
Diagonally Ordered Magic Squares

- “Streamlined Constraint Reasoning”
  - Gomes and Sellmann, CP 2004
  - Uses restarts and “streamlining” (search space restriction)
- “Disco Novo Gogo”
  - Sellmann and Ansotegui, AAAI 2006
  - Uses restarts, randomized variable ordering and learning on the value selection heuristic
- I wanted to see how CP Optimizer's search compared

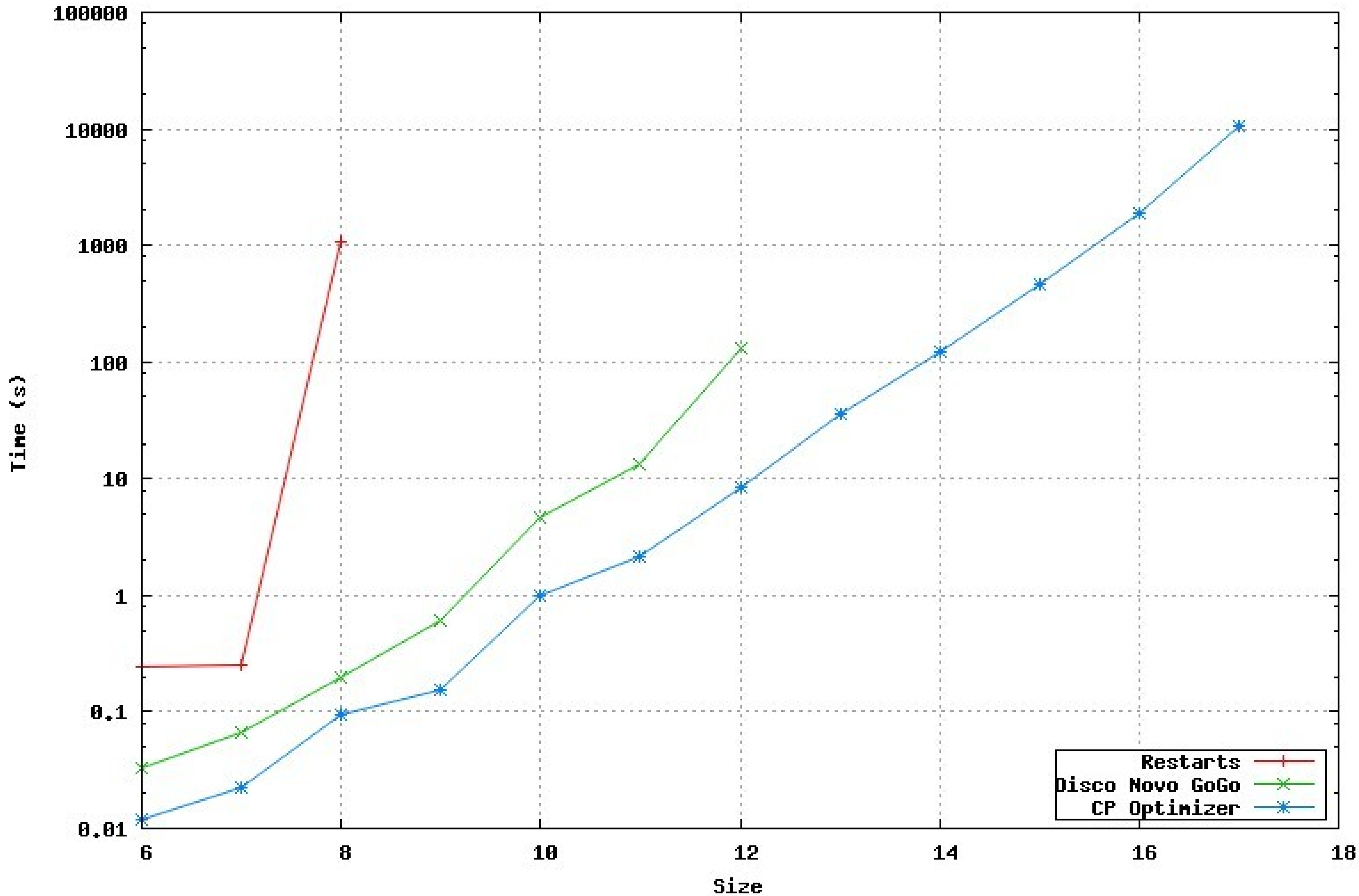
Diagonally Ordered Magic Squares



Diagonally Ordered Magic Squares



Diagonally Ordered Magic Squares



Streamlining: Dumbledore Squares

- Each row and column usually has an even spread of numbers

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

Streamlining: Dumbledore Squares

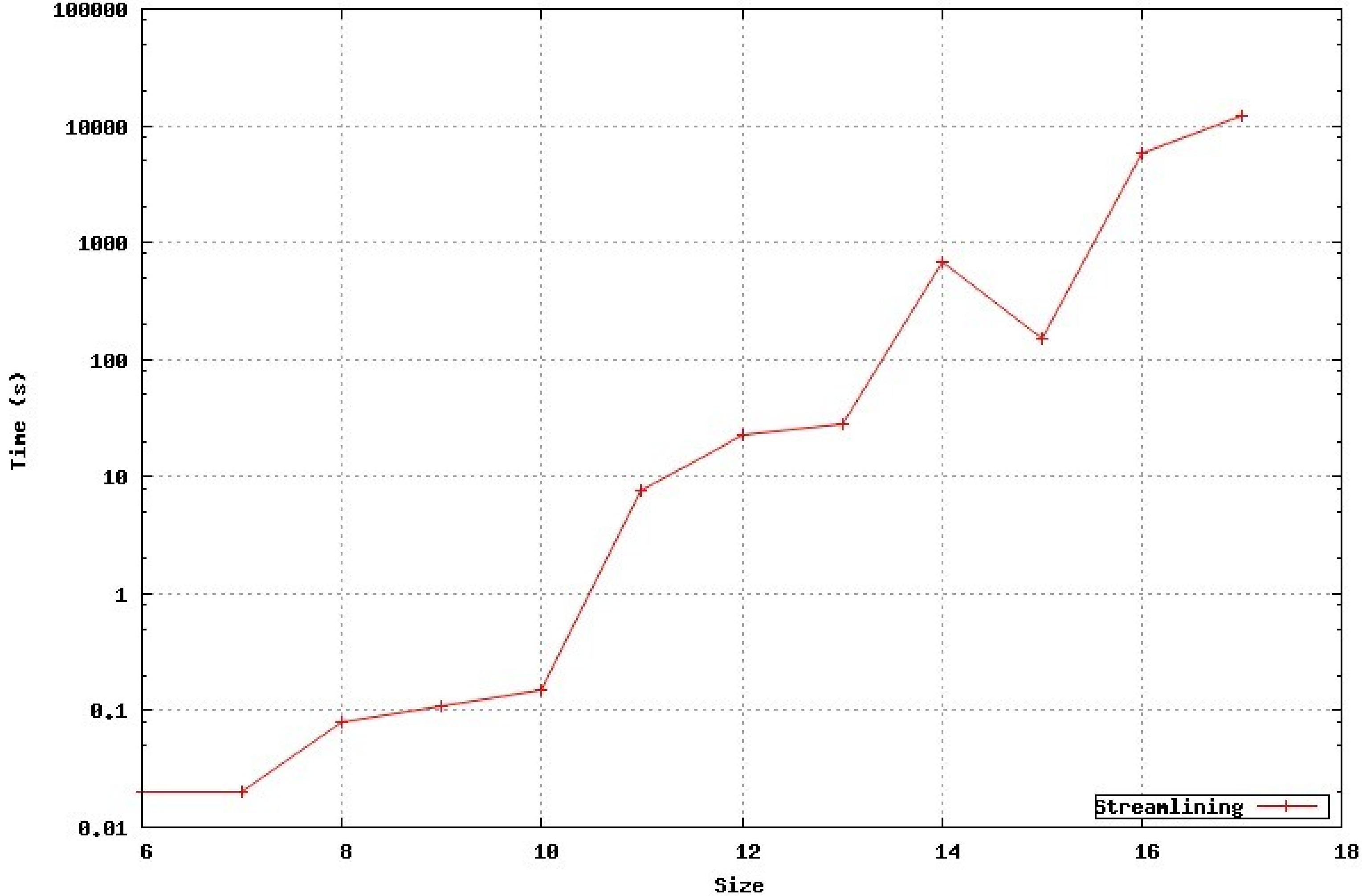
- Force each row and column to have a number from each "class"

7	18	25	4	11
5	8	19	12	21
16	24	13	9	3
22	14	2	17	10
15	1	6	23	20

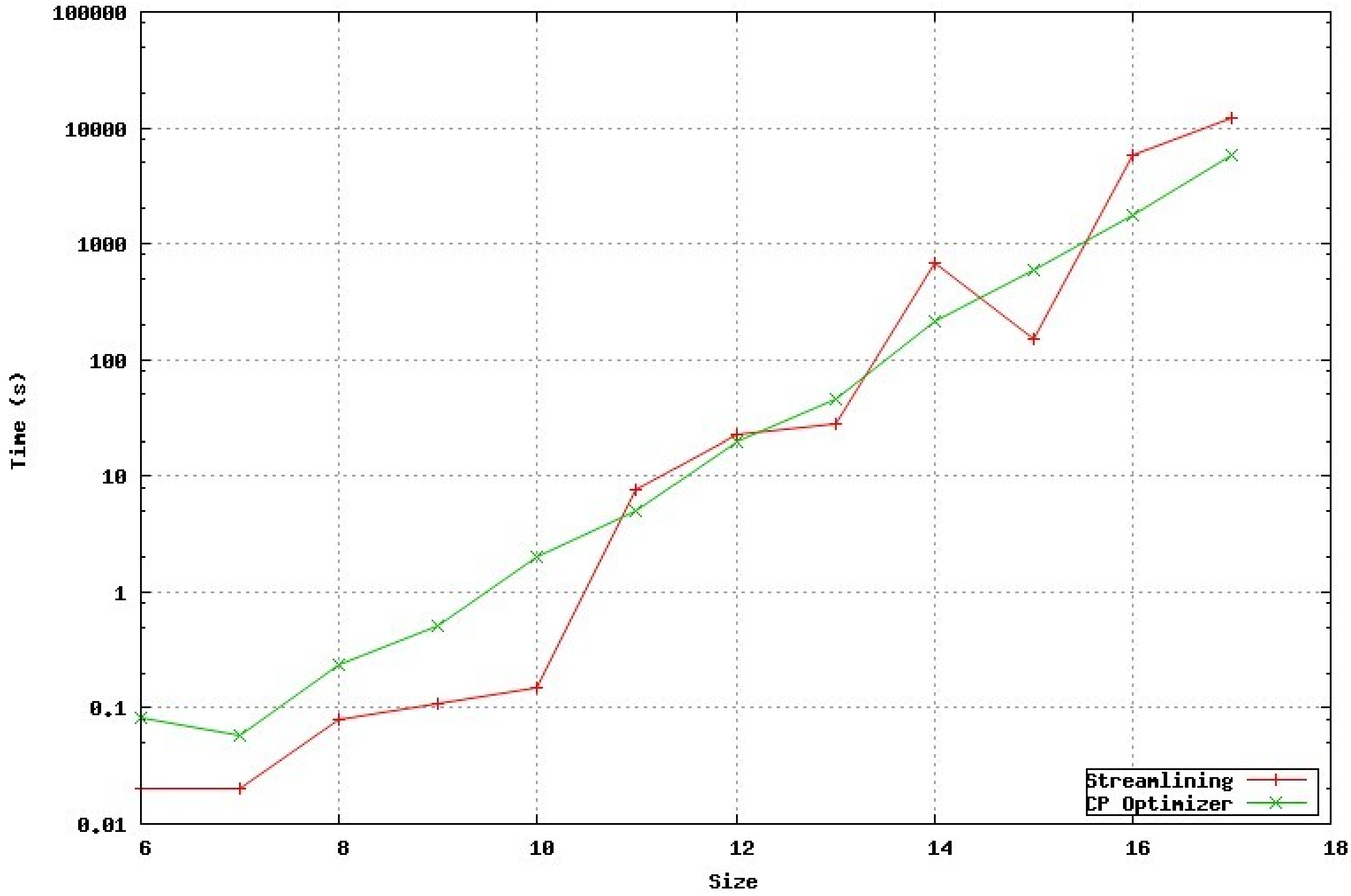
B	D	E	A	C
A	B	D	C	E
D	E	C	B	A
E	C	A	D	B
C	A	B	E	D

(A) 1-5 (B) 6-10 (C) 11-15 (D) 16-20 (E) 21-25

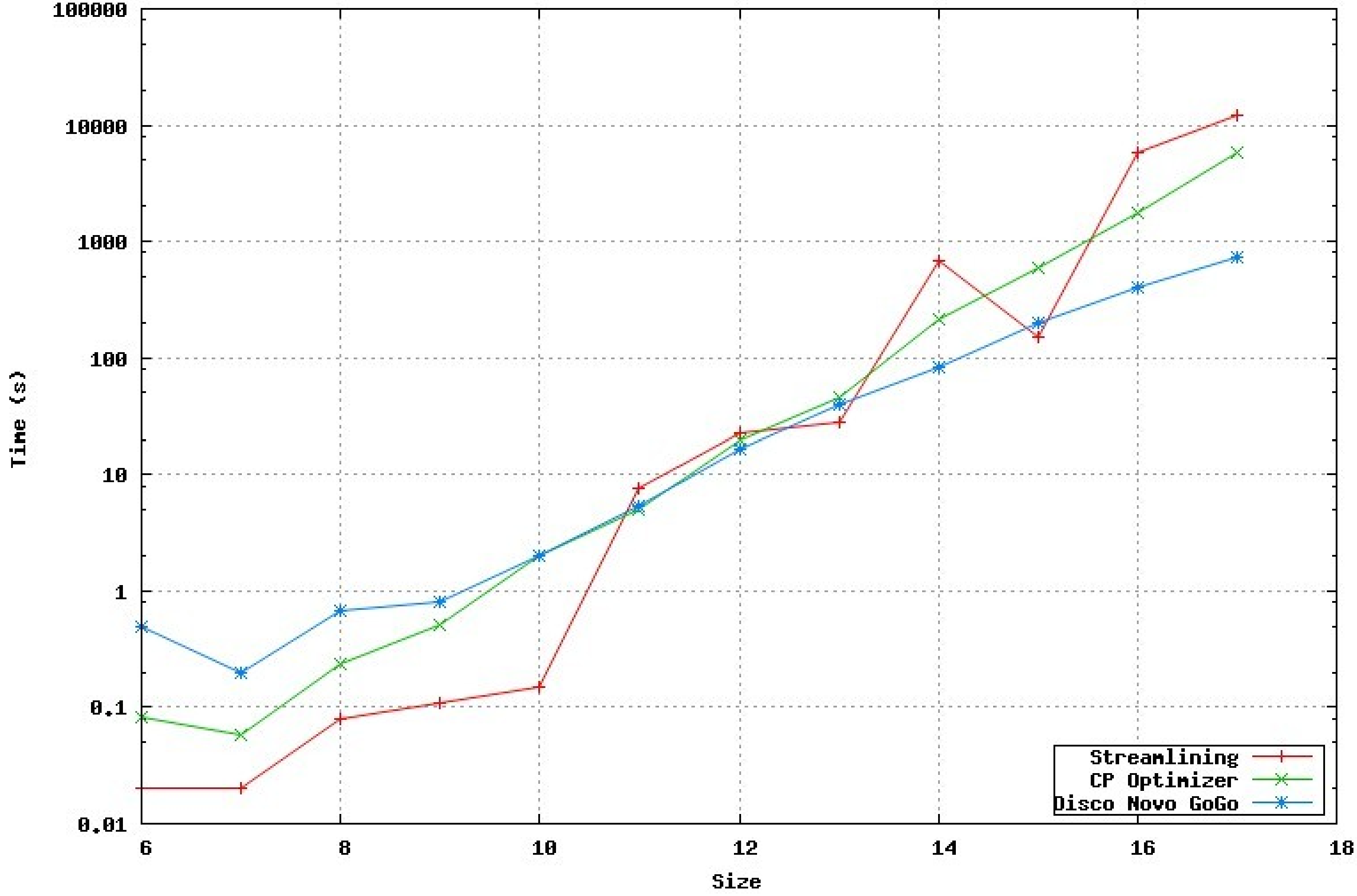
Streamlining: Dumbledore Squares



Streamlining: Dumbledore Squares



Streamlining: Dumbledore Squares



DO FAVOUR GRAPHS OVER TABLES

DO FAVOUR GRAPHS OVER TABLES

- DO produce scaling results when you can
  - These give excellent information about how different methods compare

DO FAVOUR GRAPHS OVER TABLES

- DO produce scaling results when you can
  - These give excellent information about how different methods compare
- DO use scatter plots
  - When results cannot easily be aggregated

DO FAVOUR GRAPHS OVER TABLES

- DO produce scaling results when you can
  - These give excellent information about how different methods compare
- DO use scatter plots
  - When results cannot easily be aggregated
- DO convert tables you find in the literature to graphs
  - DON'T use tables just because previous papers did!

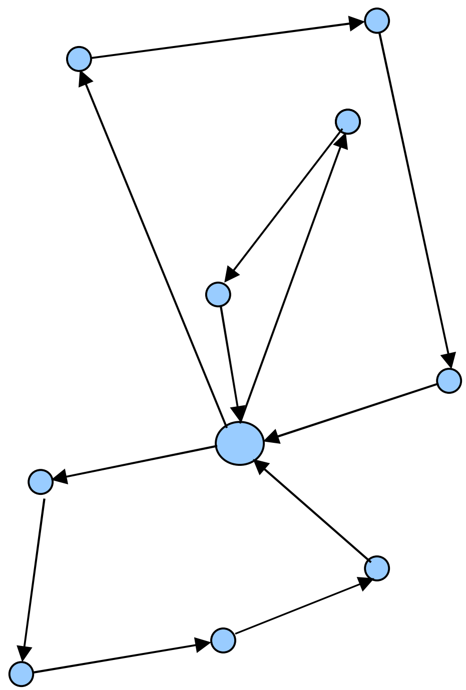
Large neighborhood search

- Hybrid of local and constructive search which looks like local search from a high level, but uses constructive search to make moves
  - Each move removes part of the current solution
  - Rebuilds it using a constructive method (usually limited in backtracks)

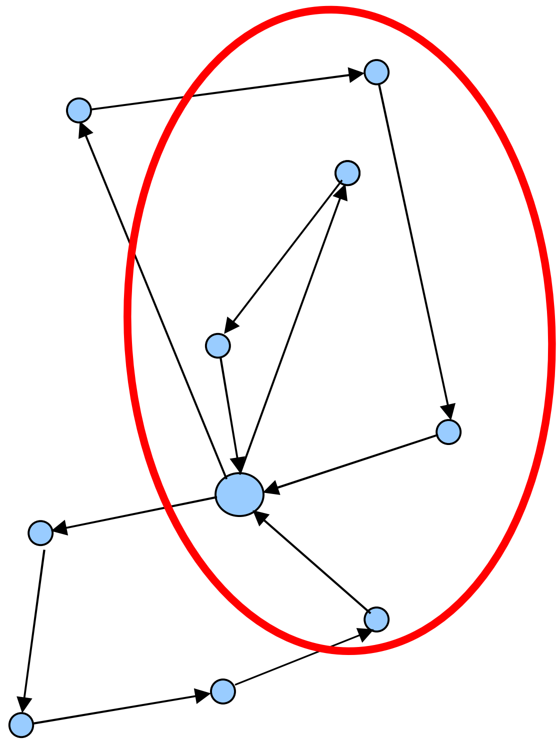
Large neighborhood search

- Hybrid of local and constructive search which looks like local search from a high level, but uses constructive search to make moves
  - Each move removes part of the current solution
  - Rebuilds it using a constructive method (usually limited in backtracks)
- I applied LNS to routing problems, and tested on the well-known “Solomon” instances of capacitated vehicle routing problems with time windows
  - This benchmark suite of 56 problems has been used in hundreds of papers on vehicle routing.
  - My LNS method removed some customers from routes, then re-inserted them using a backtracking technique and ordering heuristics

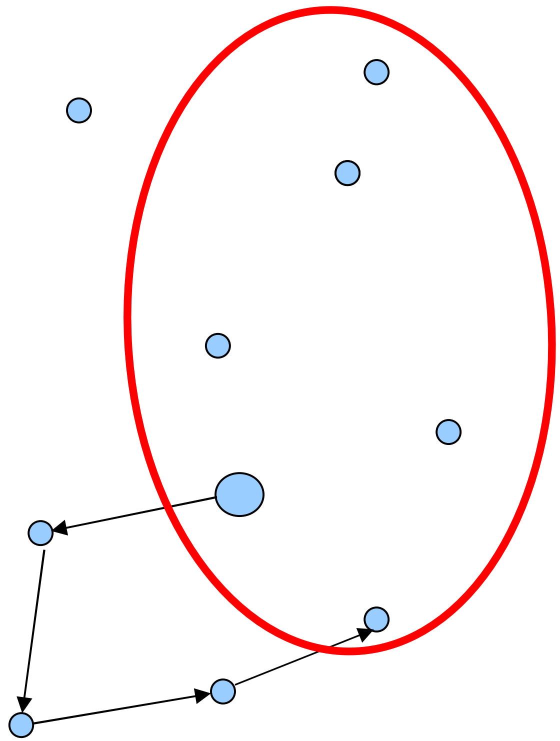
Large neighborhood search



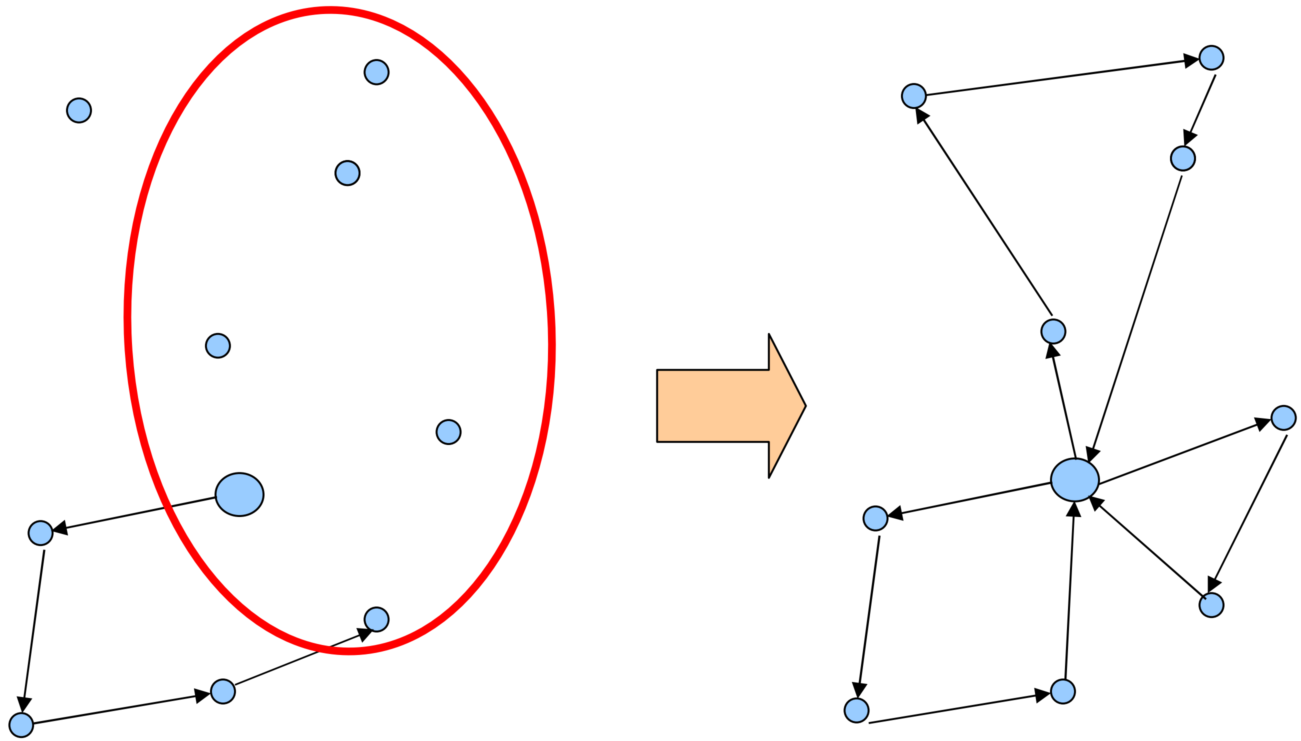
Large neighborhood search



Large neighborhood search



Large neighborhood search



Solomon problem instances (100 customers)

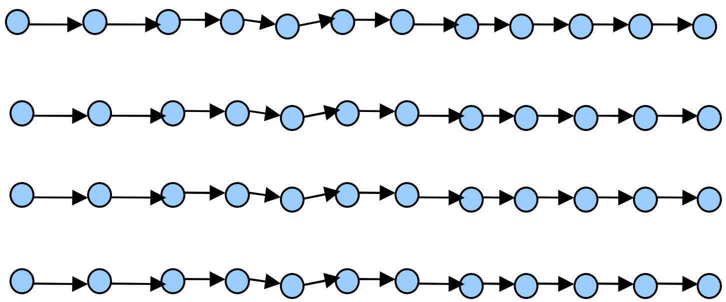
- Objective is to
  - As a primary objective, minimize the number of vehicles used
  - As a secondary objective, minimize the distance travelled
  - `obj = M * vehicles + distance`

	5-10 customers per route	25-50 customers per route
Random positions	R1	R2
Clustered positions	C1	C2
Mixed Positions	RC1	RC2

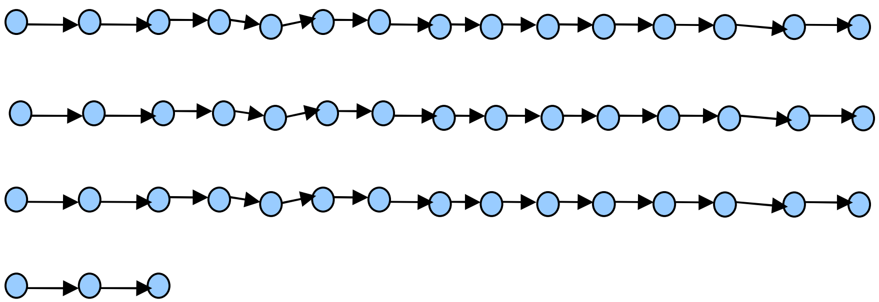
Typical situation on long-route problems (series 2)

- For the most part, LNS will reduce the distance and not the vehicles
  - To reduce the number of vehicles, LNS must remove and successfully reinsert all customers on a single vehicle
  - When average customers on a route >=12, this gets difficult

Bad for reducing vehicles



Good for reducing vehicles



Solomon problem instances (100 customers)

- Objective is to
  - As a primary objective, minimize the number of vehicles used
  - As a secondary objective, minimize the distance travelled
  - `obj = M * vehicles + distance`

	5-10 customers per route	25-50 customers per route
Random positions	R1	R2
Clustered positions	C1	C2
Mixed Positions	RC1	RC2

Solomon problem instances (100 customers)

- Objective is to
  - As a primary objective, minimize the number of vehicles used
  - As a secondary objective, minimize the distance travelled
  - `obj = M * vehicles + distance`

	5-10 customers per route	25-50 customers per route
Random positions	R1	~~R2~~
Clustered positions	C1	~~C2~~
Mixed Positions	RC1	~~RC2~~

DON'T be a slave to benchmark suites

DON'T be a slave to benchmark suites

- DON'T feel that your algorithm has to be good everywhere
  - But DO know how it performs in as many places as possible

DON'T be a slave to benchmark suites

- DON'T feel that your algorithm has to be good everywhere
  - But DO know how it performs in as many places as possible
- DO report your failures

DON'T be a slave to benchmark suites

- DON'T feel that your algorithm has to be good everywhere
  - But DO know how it performs in as many places as possible
- DO report your failures
- If you need to, DO create new benchmark instances,
  - But DO be credible

Heuristics

- Variable and value ordering heuristics are ubiquitous
- Typical implementation of first fail:

```
best = -1;
bestSize = infinity;
for i in 1..n
  if (not fixed(x[i]) and domain-size(x[i]) < bestSize)
    best = i
    bestSize = domain-size(x[i])
  end if
end for
```

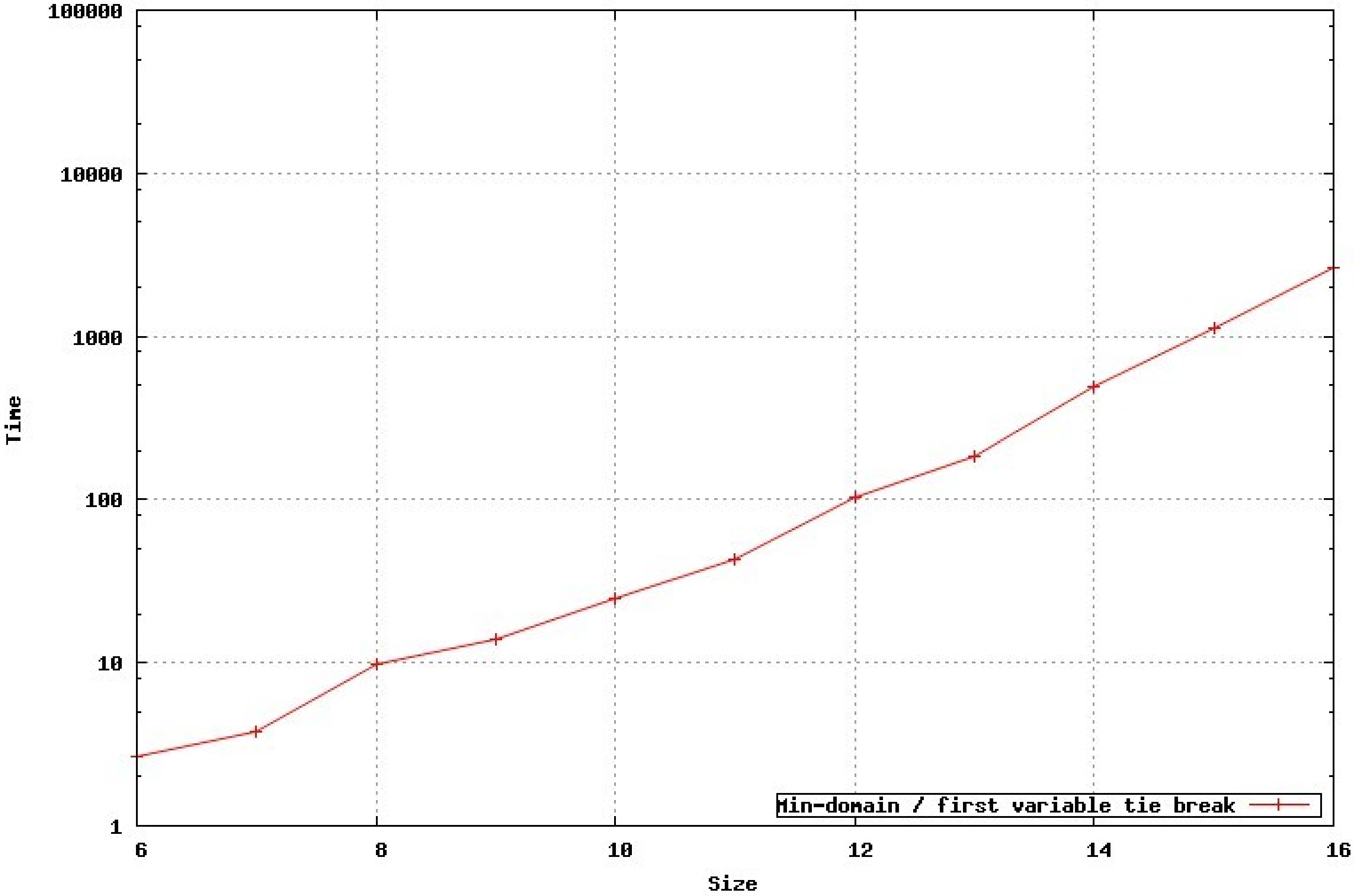
Heuristics

- Variable and value ordering heuristics are ubiquitous
- Typical implementation of first fail:

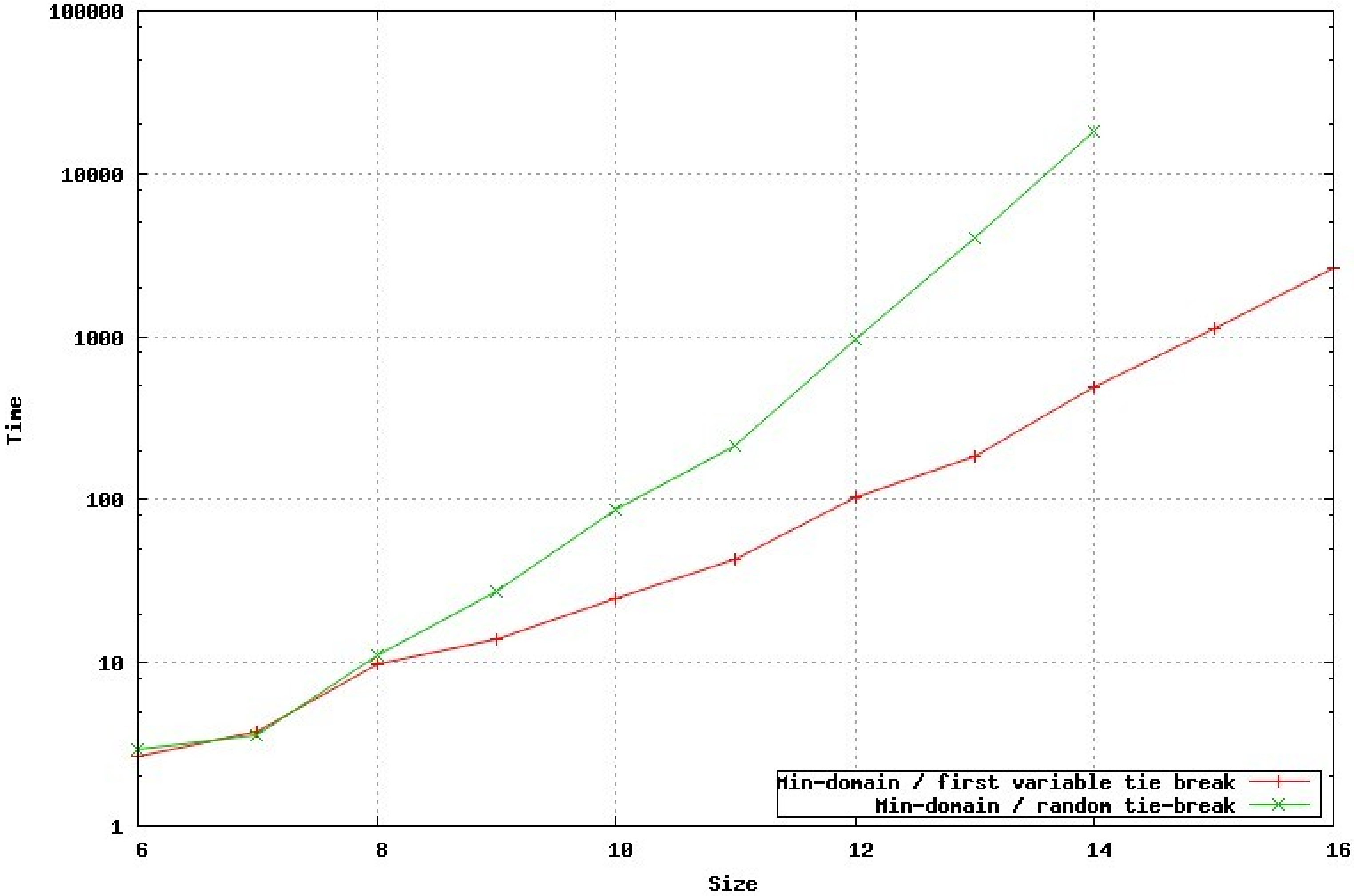
```
best = -1;
bestSize = infinity;
for i in 1..n
  if (not fixed(x[i]) and domain-size(x[i]) < bestSize)
    best = i
    bestSize = domain-size(x[i])
  end if
end for
```

  - This code contains an implicit tie-breaking rule:
    - Lower indexed variables are chosen over higher indexed ones

Heuristics: Magic squares



Heuristics: Magic squares



DO think about tie breaking

DO think about tie breaking

- DON'T tie break on arbitrary data, like an index

DO think about tie breaking

- DON'T tie break on arbitrary data, like an index
- DON'T wrongly attribute performance to the major selection rule
  - Test the minor selection rules as well

Summary

- DON'T trust yourself
  - If it looks too good to be true, then it probably is
- DON'T forget to try the obvious
  - Your “obvious” might not be the same as others'
  - The obvious might work now, when it didn't before
- DO use graphs over tables
  - Will make you ask much more interesting questions
- DON'T be a slave to benchmark suites
  - Be honest, report your failures
- DO think about tie-breaking
  - Can have a massive impact on benchmark results