DM877

Constraint Programming

Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume and Outlook

- Modeling in CP
- Global constraints (declaration)
- Notions of local consistency
- Global constraints (operational: filtering algorithms)
- Search
- Set variables
- Symmetry breaking

Search

- Complete
  - backtracking
- Incomplete
  - local search

Outline

1. Complete Search
2. Incomplete Search
3. Random Restart
4. Implementation Issues

Backtracking: Terminology

- backtracking: depth first search of a search tree
- branching strategy: method to extend a node in the tree
- node visited if generated by the algorithm
- constraint propagation prunes subtrees
- deadend: if the node does not lead to a solution
- thrashing repeated exploration of failing subtree differing only in assignments to variables irrelevant to the failure of the subtree.

Simple Backtracking

- at level j: instantiation $I = \{x_1 = a_1, \ldots, x_j = a_j\}$
- branches: different choices for an unassigned variable: $I \cup \{x = a\}$
- branching constraints $\mathcal{C} = \{b_1, \ldots, b_j\}$, $b_i, 1 \leq i \leq j$
- $\mathcal{C} \cup \{b^1_{j+1}\}, \ldots, \mathcal{C} \cup \{b^k_{j+1}\}$ extension of a node by mutually exclusive branching constraints

(In this view, easy implementation of propagation: the branching constraints are simply scheduled for propagation)

Branching strategies

Assume a variable order and a value order (e.g., lexicographic):

A. Generic branching with unary constraints:

1. Enumeration, d-way

$$x = 1 \quad | \quad x = 2 \quad | \ldots$$

2. Binary choice points, 2-way

$$x = 1 \quad | \quad x \neq 1$$

3. Domain splitting

$$x \leq 3 \quad | \quad x > 3$$

⇝ d-way can be simulated by 2-way with no loss of efficiency. While, theoretical studies (eg, [Hwang and Mitchell, 2005]) show that the viceversa is less efficient.

⇝ in practice 2-way seems more efficient than d-way on the same models

Branching strategies

B. Problem specific:

- Disjunctive scheduling (job-shop scheduling)
  s_i, s_j starting times of activities, d_i their duration
  on a shared resource:

  $$s_i + d_i \leq s_j \quad | \quad s_j + d_j \leq s_i$$

  equivalent to:

  $$x_{ij} = 1 \quad | \quad x_{ij} \neq 1$$

  with $x_{ij} = 1 \iff s_i + d_i \leq s_j$ and $x_{ij} = 0 \iff s_j + d_j \leq s_i$
  introducing binary variables for order.

- Zykov's branching rule for graph coloring

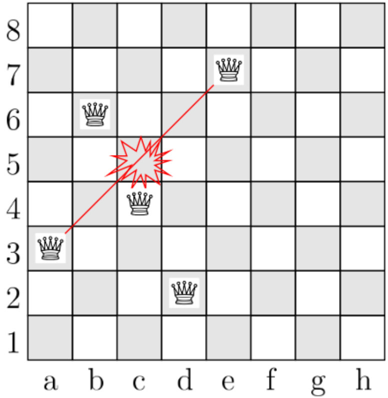
Constraint propagation

- Constraint propagation performed at each node: mechanism to avoid thrashing
- typically best to enforce domain consistency but with some exceptions (e.g., forward checking is best in SAT)
- nogood constraints added after deadend is encountered
  similar to caching or memoization techniques: record solution to subproblems and reuse them instead of recomputing them.
  Corresponds to values ruled out by higher order consistency which would be too costly to check again

Nogood constraints
Recollection

- An instantiation I on $\mathcal{P}$ is globally consistent if it can be extended to a solution of $\mathcal{P}$, globally inconsistent otherwise.
- A globally inconsistent instantiation is also called a (standard) nogood.
(a partial instantiation that does not lead to a solution.)
- Remark: A locally inconsistent instantiation is a nogood. The reverse is not necessarily true

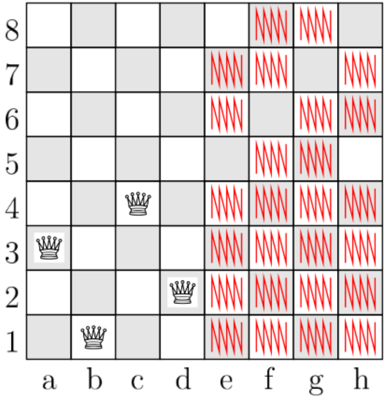
Example



$\{(x_a,3),(x_b,6),(x_c,4),(x_d,2),(x_e,7)\}$ is locally inconsistent

☞ this is a nogood.

Example



$\{(x_a,3),(x_b,1),(x_c,4),(x_d,2)\}$ is globally inconsistent

☞ this is a nogood.

Nogood constraints

Definition (Nogood)

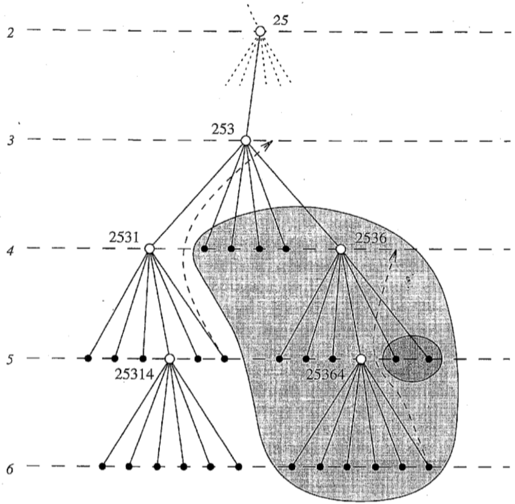
A nogood constraint is a set of assignments and branching constraints that is not consistent with any solution.

Implicit constraints, their addition does not remove solutions. Goal: reduce thrashing.

- Rule out inconsistencies before they are encountered during search:
  - Add implied constraints by hand during modeling
  - Automatically add them by applying constraint propagation algorithms

$\rightsquigarrow$ Rule out inconsistencies after they have been encountered
late for this node, since it has been already refuted, but it may contribute to pruning in the future.

E.g.: On 6-queens problem:



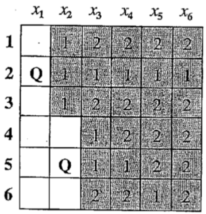
Tree on the left:

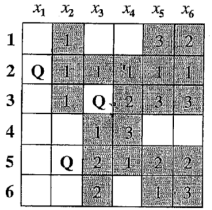
No constraint propagation

white nodes: all constraints with some instantiated variables are satisfied
black nodes: one or more constraint checks fail

shaded area explained later

With arc consistency (left) and forward checking (right):





- $\{x_1 = 2, x_2 = 5, x_3 = 3\}$ is a nogood: post $\neg\{x_1 = 2 \wedge x_2 = 5 \wedge x_3 = 3\}$
  - $\{x_1 = 5, x_2 = 2, x_3 = 4\}$ is also a nogood by mirroring over x-axis
- $(x_2 = 5) \implies (x_6 \neq 1)$ this is also a nogood.

Discovering nogoods

Case without propagation:

- Let $\mathcal{P} = \langle X, \mathcal{D}, \mathcal{C} \cup \{b_1 \dots, b_j\} \rangle$ be a deadended node (b_i, $1 \leq i \leq j$, is the branching constraint posted at level i in the search tree).
- $J(\mathcal{P})$ jumpback nogood for $\mathcal{P}$ is defined recursively:
  - $\mathcal{P}$ is a leaf node. Let C be a constraint that is not consistent with $\mathcal{P}$:

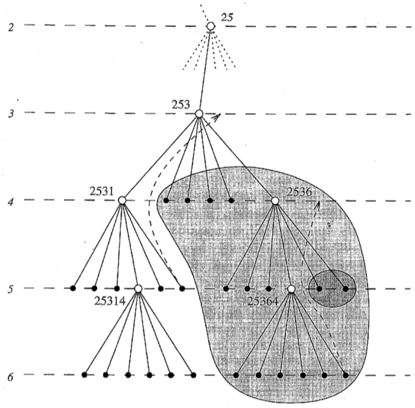
$$J(\mathcal{P}) = \{b_i \mid X(b_i) \cap X(C) \neq \emptyset, 1 \leq i \leq j\}$$

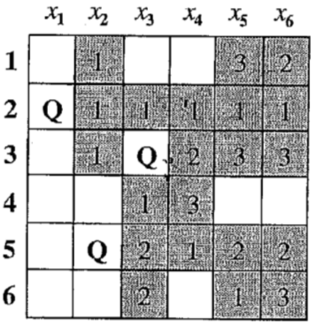
  - $\mathcal{P}$ is not a leaf node. Let $\{b_{j+1}^1 \dots, b_{j+1}^k\}$ be all possible extensions of $\mathcal{P}$ attempted by the branching strategy, each of which has failed:

$$J(\mathcal{P}) = \bigcup_{i=1}^{k} \left(J(\mathcal{P} \cup \{b_{j+1}^i\}) - \{b_{j+1}^i\} \right)$$

Example

Assume no constraint propagation





Eg: $\mathcal{C}' = \mathcal{C} \cup \{x_1 = 2, x_2 = 5, x_3 = 3, x_4 = 1, x_5 = 4\}$, all extensions of x_6 to $\mathcal{P}$ fail:

$$
\begin{aligned}
J(\mathcal{P}) &= (J(\mathcal{P} \cup \{x_6 = 1\}) - \{x_6 = 1\}) \cup \ldots \cup (J(\mathcal{P} \cup \{x_6 = 6\}) - \{x_6 = 6\}) \\
&= \{x_2 = 5\} \cup \{x_1 = 2\} \cup \{x_3 = 3\} \cup \{x_5 = 4\} \cup \{x_2 = 5\} \cup \{x_3 = 3\} \\
&= \{x_1 = 2, x_2 = 5, x_3 = 3, x_5 = 4\}
\end{aligned}
$$

Discovering nogoods

Case with propagation:

- Let $\mathcal{P} = \langle X, \mathcal{D}, \mathcal{C} \cup \{b_1 \ldots, b_j\}\rangle$ be a deadended node (b_i, $1 \leq i \leq j$, is the branching constraint posted at level i in the search tree).

- $J(\mathcal{P})$ jumpback nogood for $\mathcal{P}$ is defined recursively:
  - $\mathcal{P}$ is a leaf node. Let x be a variable whose domain has become empty (one must exist), where $\mathrm{dom}(x)$ is the original domain of x:

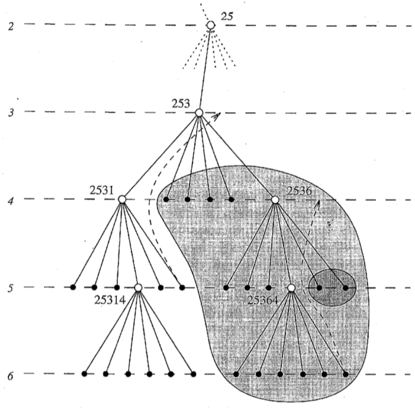
    $$J(\mathcal{P}) = \bigcup_{a \in \mathrm{dom}(x)} \mathsf{expl}(x \neq a)$$

    expl is eliminating explanation for a, ie, a subset of $\mathcal{C}$ such that $\mathsf{expl}(x \neq a) \cup \{x = a\}$ is a nogood

  - $\mathcal{P}$ is not a leaf node. Let $\{b^1_{j+1} \ldots, b^k_{j+1}\}$ be all possible extensions of $\mathcal{P}$ attempted by the branching strategy, each of which has failed:

    $$J(\mathcal{P}) = \bigcup_{i=1}^{k} (J(\mathcal{P} \cup \{b^i_{j+1}\}) - \{b^i_{j+1}\})$$

Example



Assume constraint propagation

	x_1	x_2	x_3	x_4	x_5	x_6
1		1	2	2	2	2
2	Q	1	1	1	1	1
3		1	2	2	2	2
4			1	2	2	2
5		Q	1	1	2	2
6			2	2	1	2

At node $\mathcal{P} = \{x_1 = 2, x_2 = 5\}$ 1 is removed from $D(x_6)$.
Eliminating explanation: $expl(x_6 \neq 1) = \{x_2 = 5\}$ ($\equiv \{x_2 = 5, x_6 = 1\}$ is a nogood)
Implied constraint $\neg(x_2 = 5 \wedge x_6 = 1) \rightsquigarrow (x_2 = 5) \implies (x_6 \neq 1)$

$expl(x_6 \neq 3) = \{x_1 = 2, x_2 = 5\} \rightsquigarrow (x_1 = 2 \wedge x_2 = 5) \implies (x_6 \neq 3)$

Nogood Databases

- Memory problems
- Attempt to restrict to only those that are useful:
  - restrict the nogood that are discovered
  - restrict the nogoods kept over time

Backjumping

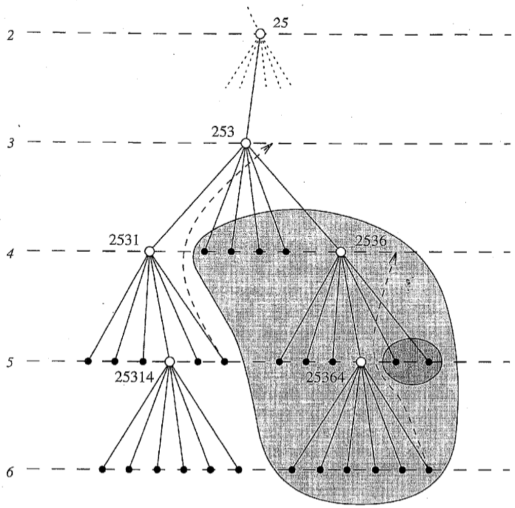
- Standard backtracking: chronological backtracking: backjump to the most recently instantiated variable
- Non-chronological backtracking $\equiv$ backjumping or intelligent backtracking: backtracks to and retracts the closest branching constraint that bears responsibility.

  Eg: jump back to the most recent variable that shares a constraint with deadend variable.

  Eg: $\mathcal{P} = \langle X, \mathcal{D}, \mathcal{C} \cup \{b_1 \ldots, b_j\}\rangle$ non-leaf deadend
  $J(\mathcal{P}) \subseteq \{b_1 \ldots, b_j\}$ jumpback nogood for $\mathcal{P}$
  jump back to largest i, $1 \leq i \leq j : b_i \in J(\mathcal{P})$ and retract b_i, all branching constraints posted after b_i and nogoods recorded after b_i

Conflict-directed backjumping

Example



- deadend after failing to extend 25314. Nogood associated is $\{x_1 = 2, x_2 = 5, x_3 = 3, x_5 = 4\}$
- Backjump to and retract $x_5 = 4$ (here like chronological backtr.)
- deadend discovered for 2531. Nogood associated is $\{x_1 = 2, x_2 = 5, x_3 = 3\}$
- backjump to and retract $x_3 = 3$ (dashed arrow) $\rightsquigarrow$ skip all the shaded tree
- (nogood used only to backjump not for propagation, less memory usage)

Restoration Service

What do we have at the nodes of the search tree?
A computational space:

1. Partial assignments of values to variables
2. Unassigned variables
3. Suspended propagators

How to restore when backtracking?

- Trailing Changes to nodes are recorded such that they can be undone later
- Copying A copy of a node is created before the node is changed
- Recomputation If needed, a node is recomputed from root

- Having more than a single node available for exploration is essential to search strategies like concurrent, parallel, or breadth-first.
- Combine recomputation with copying and trailing:
  - copy (or start trailing) a node from time to time during exploration.
  - recomputation then can start from the last copied (or trailed) node on the path to the root.
- Adaptive recomputation: as soon as a failed node occurs during exploration, the attitude for further exploration should become more pessimistic $\rightsquigarrow$ during recomputation an additional copy is created at the middle of the path for recomputation

Exploration Heuristics

Decisions must be made on Variable-Value ordering:
optimal strategy if it visits the fewest number of nodes in the search tree.
Finding optimal ordering is hard

Possible goals

- Minimize the underlying search space
- Minimize expected depth of any branch
- Minimize expected number of branches
- Minimize size of search space explored by backtracking algorithm
  (intractable to find "best" variable)

dynamic vs static strategy
In Gecode: Variable-Value Branching ch. 8 +
http://www.gecode.org/doc-latest/reference/group__TaskModelIntBranchVar.html

Variable ordering

dynamic heuristics:

- dom: choose x that minimizes $\text{rem}(x|\mathcal{P})$ the domain size remaining after propagation and branching constraints up to $\mathcal{P}$.
- $\text{dom} + \text{deg}$ (# constraints that involve a variable still unassigned)
- $\frac{\text{dom}}{\text{wdeg}}$ weight incremented when a constraint is responsible for a deadend
- min regret
  difference between smallest and second smallest value still in the domain
- structure guided var ordering:
  instantiate first variables that decompose the constraint graph
  graph separators: subset of vertices or edges that when removed separates the graph into disjoint subcomponents

Value ordering

- estimate number of solutions:
  counting solutions to a problem with tree structure can be done in polytime
  reduce the graph to a tree by dropping constraints
- if optimization constraints: reduced cost to rank values

Best First Search

- If problem unsatisfiable then DFS is the best way to go
- If problem satisfiable then BFS Best First Search is better

Variants to best search

- Limited Discrepancy search

  Discrepancy: when the search does not follow the value ordering heuristic and does not take the left most branch out of a node.

  explored tree by iteratively increasing number of discrepancies, preferring discrepancies near the root
  (thus easier to recover from early mistakes)

  Ex: ith iteration: visit all leaf nodes up to i discrepancies
  $i = 0, 1, \ldots, k$ (if $k \geq n$ depth then alg is complete)

- Interleaved depth first search
  each subtree rooted at a branch is searched for a given time-slice using depth-first.
  If no solution found, search suspended, next branch active.
  Upon suspending in the last the first again becomes active.
  Similar idea in credit based.

Randomization in Search Tree

- Dynamical selection of solution components
  in construction or choice points in backtracking.

- Randomization of construction method or
  selection of choice points in backtracking
  while still maintaining the method complete
  $\rightsquigarrow$ *randomized systematic search*.

- do backtracking until distance from a deadend has exceeded a fixed cutoff number, restart by reordering the variables

- Randomization can also be used in incomplete search

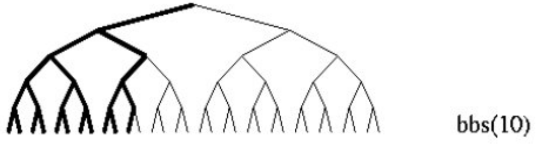
Optimization

- Solve a sequence of CSPs:
  - iterating from smallest value in domain of *cost* to largest until a solution is found
  - iterating from largest to smallest until a solution is no longer found
  - performing binary search
- use constraint propagation techniques for objective constraints

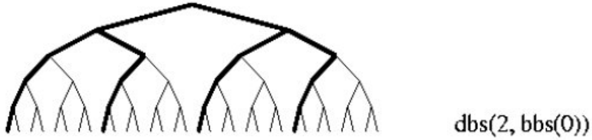
Outline

Incomplete Search

Bounded-backtrack search:



Depth-bounded, then bounded-backtrack search:

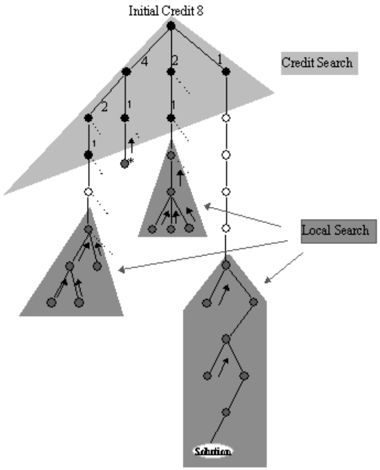


http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm

Incomplete Search

Credit-based search

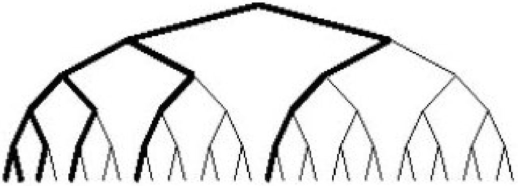
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, distribution of credit among the children, amount of local backtracking at bottom.



Incomplete Search

Limited Discrepancy Search (LDS)

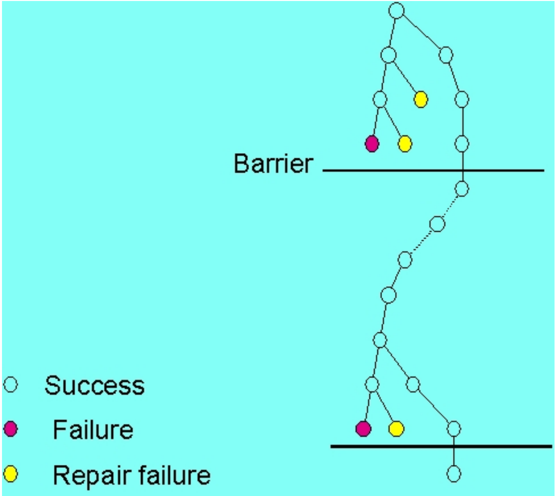
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Incomplete Search

Barrier Search

- Extension of LDS
- Key idea: we may encounter several, independent problems in our heuristic choice. Each of these problems can be overcome locally with a limited amount of backtracking.
- At each barrier start LDS-based backtracking



Outline

1. Complete Search
2. Incomplete Search
3. Random Restart
4. Implementation Issues

Randomization in Search Tree

- Ordering heuristics make mistakes (possibly early) $\rightsquigarrow$ randomization and restarts
- Randomization of choice points in backtracking
  while still maintaining the method complete
  $\rightsquigarrow$ *randomized systematic search*.
- do backtracking until distance from a deadend has exceeded a fixed cutoff number, restart by reordering the variables

Motivations

Definition (Las Vegas algorithms)

Las Vegas algorithms are randomized algorithms that always give the correct answer when they terminate, but running time varies from one run to another and is modeled as a random variable

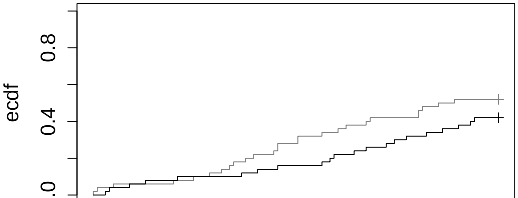
Algorithm Survival Analysis

Run time distributions

- $T \in [0, \infty]$ time to find a solution on an instance
- $F(t) = \Pr\{T \leq t\} \quad F : [0, \infty] \mapsto [0, 1]$ cdf/RTD: Run Time Distribution
- $f(t) = \frac{dF(t)}{dt}$ pdf
- $S(t) = \Pr\{T > t\} = 1 - F(t)$ survival function
- $E[T] = \int_0^\infty tf(t)dt = \int_0^1 tdF(t) = \int_0^\infty S(t)dt$ expected run time

Empirical Comparisons

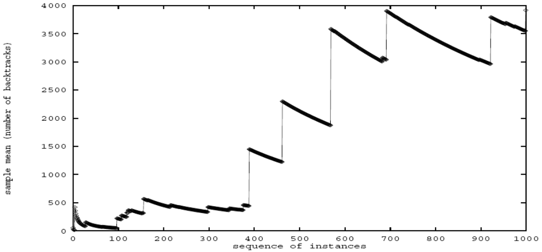
```
> load("Data/r37.RData")
> head(R37)
    time   iter event case
1    101 185737     0    1
2     57  84850     1    1
3      1    568     1    1
4     51  94974     1    1
5      5   7017     1    1

> require(survival)
> t <- survfit(Surv(time, event) ~ case, data = R37, type = "kaplan-meier",
conf.type = "plain", conf.int = 0.95, se.fit = T)
> plot(t, conf.int = F, xlab = "Time to find a solution", col = c("grey50", "black"), lty = c(1, 1), ylab = "
     ecdf", fun = "event", ylim = c(0,1))
```

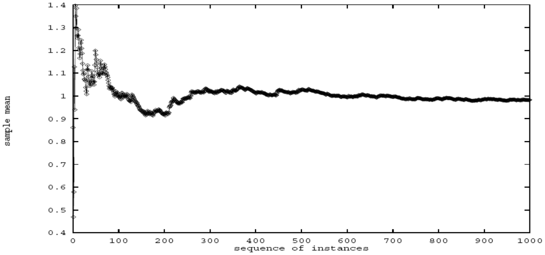


Characterization of Run-time
Heavy Tails

Gomes et al. [2000] analyze the mean computational cost to find a solution on a single instance

On the left, the observed behavior calculated over an increasing number of runs.
On the right, the case of data drawn from normal or gamma distributions





- The use of the median instead of the mean is recommended
- The existence of the moments (*e.g.*, mean, variance) is determined by the tails behavior: a case like the left one arises in presence of long tails

Heavy Tails

Standard pdf, eg the normal distribution, have exponentially decreasing tails, ie, events that are several standard deviations from the mean of the distribution are very rare.

Power law decay:

$$F(t) \underset{t \to \infty}{\longrightarrow} 1 - Ct^{-\frac{1}{\gamma}} \qquad \text{(Pareto like distr.)}$$

where $\gamma > 0$ and $C > 0$ are constants.

- Depending on C, γ, the mean of a heavy-tail distribution can be finite or not, while higher moments are infinite.

Why can RTDs have heavy tails?
Because heuristics make mistakes which require the backtracking algorithm to explore a large subtree with no solutions.

- Value mistake: a node in the search tree that is a nogood but the parent of the node is not a nogood.
- Backdoor mistake: a selection of a variable that is not in a minimal backdoor, when such a variable is available to be chosen.
  Backdoors are set of variables that if instantiated make the subproblem much easier to solve (polynomially)

Characterization of runtime

Parametric models used in the analysis of run-times to
exploit the properties of the model (eg, the character of tails and completion rate)

Procedure:

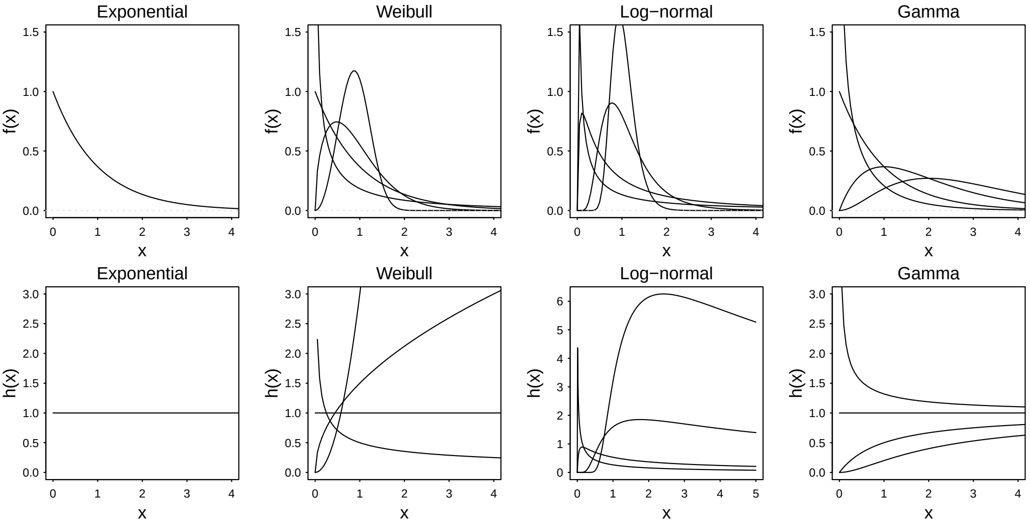
- choose a model
- apply fitting method
  maximum likelihood estimation method:

$$\max_{\theta \in \Theta} \log \prod_{i=1}^{n} p(X_i, \theta)$$

- test the model

Parametric models

The distributions used are [Frost et al., 1997; Gomes et al., 2000]:



Characterization of Run-time

Motivations for these distributions:

- qualitative information on the completion rate (= hazard function)
- empirical good fitting

To check whether a parametric family of models is reasonable the idea is to make plots that should be linear. Departures from linearity of the data can be easily appreciated by eye.

Example: for an exponential distribution:

$$\log S(t) = -\lambda t \qquad S(t) = 1 - F(t) \text{ is the survivor function}$$

$\rightsquigarrow$ the plot of $\log S(t)$ against t should be linear.

Similarly, for the Weibull the cumulative hazard function is linear on a log-log plot

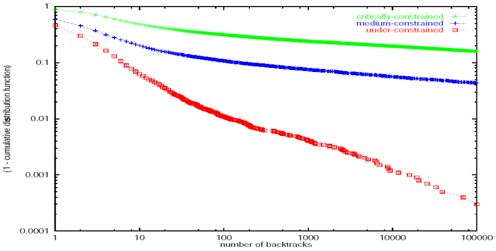
$\rightsquigarrow$ heavy tail if $S(t)$ in log-log plot is linear with slope $-1/\gamma$

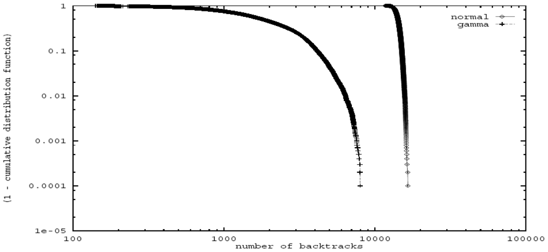
Characterization of Run-time

Heavy Tails

Graphical check using a log-log plot:

- heavy tail distributions approximate linear decay,
- exponentially decreasing tail has faster-than linear decay





Long tails explain the goodness of random restart. Determining the cutoff time is however not trivial.

Extreme Value Statistics

- Extreme value statistics focuses on characteristics related to the tails of a distribution function
  1. extreme quantiles (*e.g.*, minima)
  2. indices describing tail decay

- 'Classical' statistical theory: analysis of means.
  Central limit theorem: $X_1, \ldots, X_n$ i.i.d. with F_X

$$\sqrt{n}\frac{\bar{X} - \mu}{\sqrt{Var(X)}} \xrightarrow{D} N(0, 1), \qquad \text{as } n \to \infty$$

Heavy tailed distributions: mean and/or variance may not be finite!

Extreme Value Statistics

Extreme values theory

- $X_1, X_2, \dots, X_n$ i.i.d. F_X
  Ascending order statistics $X_n^{(1)} \leq \dots \leq X_n^{(n)}$
- For the minimum $X_n^{(1)}$ it is $F_{X_n^{(1)}} = 1 - [1 - F_X^{(1)}]^n$ but not very useful in practice as F_X unknown
- Theorem of [Fisher and Tippett, 1928]:
  "almost always" the normalized extreme tends in distribution to a generalized extreme distribution (GEV) as $n \to \infty$.

In practice, the distribution of extremes is approximated by a GEV:

$$F_{X_n^{(1)}}(x) \sim \begin{cases} \exp(-1(1-\gamma\frac{x-\mu}{\sigma})^{-1/\gamma}, & 1-\gamma\frac{x-\mu}{\sigma} > 0, \gamma \neq 0 \\ \exp(-\exp(\frac{x-\mu}{\sigma})), & x \in \mathbf{R}, \gamma = 0 \end{cases}$$

Parameters estimated by simulation by repeatedly sampling k values $X_{1n}, \dots, X_{kn}$, taking the extremes $X_{kn}^{(1)}$, and fitting the distribution.
γ determines the type of distribution: Weibull, Fréchet, Gumbel, ...

Extreme Value Statistics

Tail theory

- Work with data exceeding a high threshold.
- Conditional distribution of exceedances over threshold τ

$$1 - F_\tau(y) = P(X - \tau > y \mid X > \tau) = \frac{P(X > \tau + y)}{P(X > \tau)}$$

- If the distribution of extremes tends to GEV distribution then there exists a Pareto-type function such that for some $\gamma > 0$

$$1 - F_X(x) = x^{-\frac{1}{\gamma}} \ell_F(x), \qquad x > 0,$$

with $\ell_F(x)$ a slowly varying function at infinity.

In practice, fit a function $Cx^{-\frac{1}{\gamma}}$ to the exceedances:
$Y_j = X_i - \tau$, provided $X_i > \tau$, $j = 1, \ldots, N_\tau$.
γ determines the nature of the tail

Characterization of Run-time

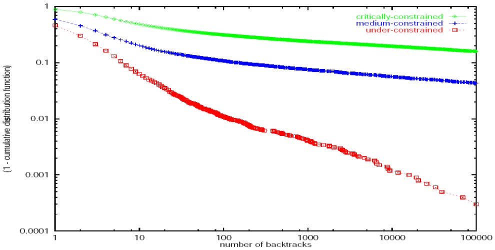
Heavy Tails

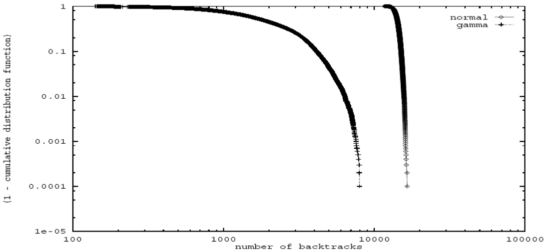
The values estimated for γ give indication on the tails:

- $\gamma > 1$: long tails hyperbolic decay (the completion rate decreases with t) and mean not finite
- $\gamma < 1$: tails exhibit exponential decay

Graphical check using a log-log plot:

- heavy tail distributions approximate linear decay,
- exponentially decreasing tail has faster-than linear decay





Long tails explain the goodness of random restart. Determining the cutoff time is however not trivial.

Randomization

- Randomize the variable ordering
- randomize tie breaking
- ranking variables within a small factor of the best variable and choosing one at random
- choose a variable with probability proportional to heuristic weight of the variable
- pick one at random from a set of heuristics to use for the selection
- randomize value ordering
- random backwards jump in search space upon backtracking (makes it incomplete)

Wanted: enough different decisions near the top of the search tree

Restart strategies

- Restart strategy: execute a sequence of runs of a randomized algorithm, to solve a single problem instance, stopping the r-th run after a time $\tau(r)$ if no solution is found, and restarting the algorithm with a different random seed

- defined by a function $\tau : \mathbb{N} \to \mathbb{R}^+$ producing the sequence of thresholds $\tau(r)$ employed.

- origins in the field of communication networks
  (Fayolle et al., 1978) derive the optimal timeout for a simple "send and wait" communication protocol, maximizing the transmission rate.

- It can be proved that restart is beneficial under two conditions: if the survival function decreases less fast than an exponential, and if the RTD is improper.

Luby et al. [1993] study Las Vegas algorithms and prove that:

- if $F(t)$ is known:
  the optimal restart strategy is uniform, i.e., $\tau(r) = \tau$, ie, $\vec{\tau} = (\tau, \tau, \tau, \tau, \ldots)$.
  Optimal cutoff time $\vec{\tau}^*$ can be evaluated minimizing the expected value of the total run-time T_τ:

$$E\{T_{\vec{\tau}}\} = \frac{\tau - \int_0^\tau F(t)dt}{F(\tau)}$$

  (of course $F(t)$ is not known in practice)

- if $F(t)$ is not known, Luby et al. [1993] suggested a universal, non-uniform restart strategy, whose cutoff sequence is composed of powers of 2:

$$\vec{\tau}^{univ} = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots)$$

$$\tau^{univ}(r) := \begin{cases} 2^{j-1} & \text{if } r = 2^j - 1; \\ \tau(r - 2^{j-1} + 1) & \text{if } 2^{j-1} \leq r < 2^j - 1 \end{cases}$$

(everytime a pair of runs of a given length is completed a run of twice that length is executed $\equiv$ when 2^{j-1} is used twice, 2^j is the next)

⇝ For all distributions $F(t)$ the performance of $\vec{\tau}^{univ}$ is bounded with high probability with respect to $E_F\{T_{\vec{\tau}^*}\}$:

$$E_F\{T_{\vec{\tau}^{univ}}\} \leq 192 E_F\{T_{\vec{\tau}^*}\}(\log E_F\{T_{\vec{\tau}^*}\} + 5)$$

and the tail decays exponentially. (Note that the result is asymptotic)

⇝ It is the best performance it can be achieved by any universal strategy up to a constant factor

Deciding the Restart Strategy in Practice

What counts for primitive operation?

- number of deadends
- distance from a deadend (keep nogoods discovered)
- number of backtracks
- number of nodes visited

For fixed cutoff, which cutoff value?

- instance dependent: hence trial and error
- safer to make larger than too small
- in practice the universal strategy seems slow as it increases too slowly, hence often scaled version: $\vec{\tau}^{univ} = (s, s, 2s, \ldots)$
- Toby Walsh proposes a geometric progression $\vec{\tau}^{g} = (1, s, s^2, \ldots)$ for $1 < s < 2$. Performs well in practice but no guarantees.
- Kautz et al. propose a Bayesian model to predict when run will go long and restart it
- optimization within a given deadline also possible

Outline

1. Complete Search

2. Incomplete Search

3. Random Restart

4. Implementation Issues

Search – Resume

- Backtracking
- Branching strategies (Variable-Value heuristics)
- Nogood constraints
- Backjumping
- Restoration service
  Gecode (sec. 9.1 + ch. 42) uses a hybrid recomputation: cloning (`copy`) + batch recomputation (`commit` + `status`)

  c_d=8 clones such that recomputation makes less than 8 commits
  In addition: adaptive recomputation, more copying when a deadend encountered
  if a node must be recomputed, an additional clone is created in the middle of the recomputation path

  a_d=2 recomputation adaptation distance (only if path length $n > a_d$ a copy is created)

In Gecode

- Branching (ch.8) defines the shape of the search tree.
- Exploration (ch.9) defines a strategy how to explore parts of the search tree

In Gecode

Branching (ch.8) defines the shape of the search tree.

- predefined variable-value branching for `branch()` function
- `INT_VAR_...`, `INT_VAL_...`, `SET_VAR_...`,`SET_VAL_...`
  `FLOAT_VAR_...`, `FLOAT_VAL_...`
  `Rnd r(1U);` uniform random numbers
- local selections: depend only on current node
  shared selections: use information that is collected during search, hence on all nodes created since branching posted:
  eg, Accumulated Failure Count (aka, weighted degree, wdeg, sec. 8.5.2)
  Activity-based: how many values have been removed from variable's domain
- Lightweight Dynamic Symmetry Breaking, see later

- In optimization `branch(home, c, INT_VAL_MIN());` will try values for c in increasing order (not good in parallel search)
- Filters:

```
static bool filter(const Space& home, IntVar y, int i) {
  return y.size() >= 4;
}
branch(home, x, ... , ... , &filter);
```

In Gecode

Exploration (ch.9) defines a strategy how to explore parts of the search tree

- Hybrid recomputation
- Parallel search (`-threads 8`): work-stealing architecture
  - initially, all work is given to a single worker for exploration, making the worker busy.
  - All other workers are initially idle, and try to steal work from a busy worker: ie, part of the search tree is given from a busy worker to an idle worker
  - non-deterministic
  - memory needed scales linearly with the number of workers used.
- Search engines `DFS, BAB`; `next(), statistics(), stopped()`

- `Search::Stop(Search::Statistics, Search::Options)`; `next()` passed to a search engine

- Restart from a modified problem:

  - AFC or activity heuristics are updated
  - different random seed
  - use different branching heuristic
  - include no-goods
  - Large Neighborhood Search: keep a randomly selected part of a previous solution.

- `RBS<DFS,Script> e(s,o);`

- Cutoff generators: `Search::Cutoff`; `operator()(), operator++()`, the first returns the current cutoff value and the second increments to the next cutoff value and returns it. Cutoff values are of type `unsigned long int`

```
Search::Cutoff* c = Search::Cutoff::luby(s); //s, scale factor; MPG p.155-156
Search::Options o;
o.cutoff = c;
RBS<DFS,Script> e(space,o);
```

- no-goods by deafult not activated in RBS.
- nogoods_limit describes to which depth limit no-goods should be extracted from the path of the search tree maintained by the search engine.

```
Search::Options o;
o.nogoods_limit = 128;
RBS<DFS,Script> e(s,o);
```

- larger values for this limit imply higher memory consumption

Search Options
from command line

member	type	meaning
threads	**double**	number of parallel threads to use
c_d	**unsigned int**	commit recomputation distance
a_d	**unsigned int**	adaptive recomputation distance
clone	**bool**	whether engine uses a clone when created
nogoods_limit	**unsigned int**	depth limit for no-good generation
stop	Search::Stop*	stop object (NULL if none)
cutoff	Search::Cutoff*	cutoff object (NULL if none)

Van Hentenryck's Videos

- COMET code
- Choose var that leaves more values for other variables
- Value oriented decision (eg, perfect squares)
- Weaker commitment, domain splitting, $>, <$
  (eg, magic squares, car sequencing)
  tends to be a better choice since fixing values less benefit from propagation from other variables (Tip. 8.2)
- Symmetry breaking vs heuristics

References

Frost D., Rish I., and Vila L. (1997). **Summarizing CSP hardness with continuous probability distributions.** In *Proceedings of AAAI/IAAI*, pp. 327–333.

Gomes C. and Selman B. (2005). **Can get satisfaction**. *Nature*, 435, pp. 751–752.

Gomes C., Selman B., Crato N., and Kautz H. (2000). **Heavy-tailed phenomena in satisfiability and constraint satisfaction problems**. *Journal of Automated Reasoning*, 24(1-2), pp. 67–100.

Luby M., Sinclair A., and Zuckerman D. (1993). **Optimal speedup of las vegas algorithms**. *Information Processing Letters*, 47(4), pp. 173–180.

Minton S., Johnston M., Philips A., and Laird P. (1992). **Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems**. *Artificial Intelligence*, 58(1-3), pp. 161–205.

Rossi F., van Beek P., and Walsh T. (eds.) (2006). **Handbook of Constraint Programming**. Elsevier.

Schulte C. and Carlsson M. (2006). **Finite domain constraint programming systems**. In Rossi et al. [2006].