LECTURE 11 & 12
CONSTRAINT-BASED LOCAL SEARCH
Constraint-based Local Search

• Problem given in CSP form:
  – a set of variables $V = \{V_1, V_2, ..., V_n\}$
  – a set of constraints $C = \{C_1, C_2, ..., C_k\}$
    i.e. arithmetic or symbolic relations over $V$

• Use Local Search to solve the constraints

• Cost function to minimize:
  number of violated constraints

• Naturally copes with Over-Constraint problems
Local Search for Constraint Solving

solving as optimization:
Objective function to minimize
e.g. number of unsatisfied constraints
Ressources

• P. Van Hentenryck and L. Michel
  Constraint-based Local Search
  MIT Press 2005

• COMET System 2.1
  http://dynadec.com/support/downloads/

• Adaptive Search Algorithm (C library)
  http://cri-dist.univ-paris1.fr/diaz/adaptive/
Adaptive Search

• Basic Idea from GSAT
  (Local Search for SAT)
  [Selman 92, Selman/Kautz 94]

• Extension: $n$-ary domains + arbitrary constraints

• Iterative repair
  at each iteration: 1 variable selected, value is changed

• min-conflict [Minton 92]

• Adaptive memory (cf. Tabu Search)

• Based on variables + constraints information
Adaptive search:
Error Functions

Constraint
e.g. \( X = Y \)

\textit{Error function} which measures how much the constraint is satisfied
\[ |X - Y| \]

Projection on each variable
Intuitively:

• Search guided by the structure of the problem
  – at the variable level
  – at the constraint level
  – constraints give structure to the problem & variables link them together

• Convergence on variables with « bad » values

• Any kind of constraint can be used (if appropriate error functions is defined)
Examples of Error Functions

\[ V_1 = V_2 \]

\[ V_1 < V_2 \]

\[ V_1 \in V_2 \]

\[ C_1 \land C_2 \]

\[ \text{alldiff} \ (V_1 \ldots V_n) \]

\[ \forall V_1 \in V_2, C (V_1) \]

\[ \text{Increasing} \ (V_1 \ldots V_n) \]

\[ | V_1 - V_2 | \]

\[ \max (0, 1 + V_1 - V_2) \]

\[ \min (V_1 - v), v \in V_2 \]

\[ \max (f_{C_1}, f_{C_2}) \]

\[ \text{Card} \ (V_i = V_k, i < k) \]

\[ \max f_{C_1} (V_1), V_1 \in V_2 \]

\[ \sum_{i < j < n} V_i - V_j \]

\[ V_i > V_j \]
Basic Algorithm

• Start with random configuration
• Repeat
  compute the errors on constraints
  combine the errors on each variable
  select the worst variable \( X \) (except if “tabu”)
  compute cost of configurations with other values of \( X \)
  if a better configuration is found, move to this config.
  else mark \( X \) tabu & move randomly
• Until
  a solution is found
  or a maximum number of iterations is reached
Generic Algorithm

- Parametrized by:
  - score function for constraints
  - Combination function for variables
  - cost function for configurations

- Tabu list
  - each var. leading to local minimum is marked and cannot be chosen for a few iterations

- Plateau(x):
  - when selected variable has no improving alternative value (but equal global cost)
Escaping Plateaux

• with prob $p$: choose a neighbor on the plateau
• with probability $1-p$: escape from plateau
  (mark current variable as Tabu and randomly choose another variable)
• empirical best value of $p \in [0.85,0.95]$
• example:
  – Magic Square
  – $p=94$
  – About 10 times speed-up!
Also ...

- (partial) Resets
  - Different from « Restarts »
  - Triggered by the number of Tabu variables
  - Then reset a given % of variables
  - Usually about 20%
  - Can be specialized for a given problem

- C library available as freeware since 2003
  
  http://cri-dist.univ-paris1.fr/diaz/adaptive
Adaptive Search Base Algorithm

Input: problem given in CSP format: some tuning parameters:
- variables $X_i$ with their domains
- constraints $C_j$ w/error functions
- function to project errors on vars
- cost function to minimize
- $TT$: # iterations a variable is frozen
- $RL$: # frozen variables triggering a reset
- $RP$: % of variables to reset
- $MI$: max. # iterations before restart
- $MR$: maximal # of restarts

Output: a solution if the CSP is satisfied or a quasi-solution of minimal cost otherwise.

1: $Restart \leftarrow 0$
2: repeat
3: \hspace{1em} $Restart \leftarrow Restart + 1$
4: \hspace{1em} $Iteration \leftarrow 0$
5: \hspace{1em} Compute a random assignment $A$ of variables in $V$
6: \hspace{1em} $Opt\_Sol \leftarrow A$
7: \hspace{1em} $Opt\_Cost \leftarrow cost(A)$
8: \hspace{1em} repeat
9: \hspace{2em} $Iteration \leftarrow Iteration + 1$
10: \hspace{2em} Compute errors of all constraints in $C$ and combine errors on each variable
11: \hspace{2em} \hspace{1em} $\triangleright$ (by considering only the constraints in which a variable appears)
12: \hspace{2em} Select the variable $X$ (not marked Tabu) with highest error
13: \hspace{2em} Evaluate costs of possible moves from $X$
14: \hspace{2em} if no improvement move exists then
15: \hspace{3em} mark $X$ as Tabu until iteration number: $Iteration + TT$
16: \hspace{3em} if the number of variables marked Tabu $\geq RL$ then
17: \hspace{4em} randomly reset $RP$ % variables in $V$ (and unmark those Tabu)
18: \hspace{3em} end if
19: \hspace{2em} else
20: \hspace{3em} Select best move and change $X$, yielding the next configuration $A'$
21: \hspace{3em} if $cost(A') < Opt\_Cost$ then
22: \hspace{4em} $Opt\_Sol \leftarrow A \leftarrow A'$
23: \hspace{4em} $Opt\_Cost \leftarrow cost(A')$
24: \hspace{3em} end if
25: \hspace{2em} end if
26: \hspace{1em} until $Opt\_Cost = 0$ (a solution is found) or $Iteration \geq MI$
27: until $Opt\_Cost = 0$ (a solution is found) or $Restart \geq MR$
28: output($Opt\_Sol, Opt\_Cost$)
N queens

- N variables
- Configurations = permutation of \{1,...,n\}
- Constraints (diagonals):
  \( \forall i \in [1,N], \forall j \in [i+1,N], \)
  \( Qi+i \neq Qj +j \)
  \( Qi-i \neq Qj -j \)

- Error : 0 if constraint satisfied
  1 if constraint violated (Qi = Qj)
- combination on vars. : sum
- cost : sum of errors
- move : swap 2 queens
Performance on N-Queens

<table>
<thead>
<tr>
<th>N</th>
<th>Adaptive Search (time in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.00</td>
</tr>
<tr>
<td>10000</td>
<td>0.52</td>
</tr>
<tr>
<td>30000</td>
<td>4.64</td>
</tr>
<tr>
<td>60000</td>
<td>19.05</td>
</tr>
<tr>
<td>100000</td>
<td>55.58</td>
</tr>
</tbody>
</table>

Surprisingly, runtime results are very stable for any given $n$.
Magic Square

• place all the numbers in \{1, 2, \ldots, N^2\} on a \(N\times N\) square s.t. sum of the numbers in all rows, columns and two diagonals are equal

• Constraints are equations of the form: \(\sum_i x_i = M\)
  (mean: \(M = N*(N+1)/2\))

• Plus one \textit{all\_different} constraint
Adaptive Search formulation

• $N^2$ Variables: permutation of $\{1, 2, \ldots, N^2\}$

• Constraints of the form: $\sum_i x_i = M$
  
  error fonction for equation: given $X=Y$ $error is X-Y$

• Combination: sum of errors
  
  (could be: sum of absolute values)

• cost: sum of absolute values

• move: swap 2 values

• *all_different* constraint is implicit
  
  because moves are swaps of values
# Magic Square with AS

<table>
<thead>
<tr>
<th>Values and Constraint errors</th>
<th>Projections on variables</th>
<th>Costs of next configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td></td>
<td>39 54 51 33</td>
</tr>
<tr>
<td>11 7 8 15 7 4</td>
<td>8 2 1 8</td>
<td>53 67 61 41</td>
</tr>
<tr>
<td>16 2 4 12 0</td>
<td>4 8 16 9</td>
<td>45 57 57 66</td>
</tr>
<tr>
<td>10 6 5 3 -10</td>
<td>6 23 21 1</td>
<td>77 43 48 41</td>
</tr>
<tr>
<td>1 14 9 13 3</td>
<td>1 2 5 9</td>
<td></td>
</tr>
<tr>
<td>4 -5 -8 9 -3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with other solvers

- Local search solver COMET (Dynatec)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Comet</th>
<th>Adaptive Search</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queens n=10000</td>
<td>24.5</td>
<td>0.52</td>
<td>47</td>
</tr>
<tr>
<td>Queens n=20000</td>
<td>96.2</td>
<td>2.16</td>
<td>44.5</td>
</tr>
<tr>
<td>Queens n=50000</td>
<td>599</td>
<td>13.88</td>
<td>43.2</td>
</tr>
<tr>
<td>Magic Square 30x30</td>
<td>56.5</td>
<td>0.34</td>
<td>166</td>
</tr>
<tr>
<td>Magic Square 40x40</td>
<td>199</td>
<td>0.53</td>
<td>375</td>
</tr>
<tr>
<td>Magic Square 50x50</td>
<td>609</td>
<td>1.18</td>
<td>516</td>
</tr>
</tbody>
</table>
All-Intervals series

• Find a permutation $\sigma$ of $\{1, \ldots, n\}$ s.t.
  
  $|\sigma_{i+1} - \sigma_i|$ is a permutation of $\{1, \ldots, n-1\}$

• In fact, this problem comes from musical exercises:
  
  – For $n=12$, one has musical series
    
    (all notes of the chromatic scale appear exactly once)
    where intervals between two consecutive notes are all different

  
  \[
  \begin{align*}
  &3 &3 &6 &6 &0 &7 &7 &5 &2 &2 &4 &1 &5 &4 &1 \\
  \end{align*}
  \]

• Constraints:

  \[
  \text{all\_different}(\{ |\sigma_{i+1} - \sigma_i| , i \in [1,n-1] \})
  \]

• A solution for $n=8$: 

  \[
  \begin{align*}
  &3 &3 &6 &6 &0 &7 &7 &5 &2 &2 &4 &1 &5 &4 &1 \\
  \end{align*}
  \]
Performance on All-Interval Series

<table>
<thead>
<tr>
<th>N</th>
<th>Adaptive Search (time in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.22</td>
</tr>
<tr>
<td>150</td>
<td>1.83</td>
</tr>
<tr>
<td>200</td>
<td>5.87</td>
</tr>
<tr>
<td>300</td>
<td>23.53</td>
</tr>
</tbody>
</table>
Perfect Squares

<table>
<thead>
<tr>
<th>problem instance</th>
<th>master square size</th>
<th>squares to place number</th>
<th>largest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$112 \times 112$</td>
<td>21</td>
<td>$50 \times 50$</td>
</tr>
<tr>
<td>2</td>
<td>$228 \times 228$</td>
<td>23</td>
<td>$99 \times 99$</td>
</tr>
<tr>
<td>3</td>
<td>$326 \times 326$</td>
<td>24</td>
<td>$142 \times 142$</td>
</tr>
<tr>
<td>4</td>
<td>$479 \times 479$</td>
<td>24</td>
<td>$175 \times 175$</td>
</tr>
<tr>
<td>5</td>
<td>$524 \times 524$</td>
<td>25</td>
<td>$220 \times 220$</td>
</tr>
</tbody>
</table>
Performance on Perfect Squares

<table>
<thead>
<tr>
<th>N</th>
<th>Adaptive Search (time in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.16</td>
</tr>
<tr>
<td>2</td>
<td>4.87</td>
</tr>
<tr>
<td>3</td>
<td>6.30</td>
</tr>
<tr>
<td>4</td>
<td>6.42</td>
</tr>
<tr>
<td>5</td>
<td>22.35</td>
</tr>
</tbody>
</table>
How to do Performance Analysis

• The execution time of LS algorithms (e.g. AS) varies from one run to another
• Difficult to analyze performance and compare
• A simple thing is to consider mean values
  – Also can add min, max and standard deviation
• But is it enough?

• In fact larger class of algorithms:
  – Las Vegas algorithms
Las Vegas Algorithms

• Proposed by [Babai 79] as extension of Monte Carlo algorithms
• Any algorithm for which the execution time varies for different runs on the same input:
  – Local Search algorithms & Metaheuristics
  – Quicksort with pivot element chosen at random
  – Randomized algorithms

Definition 1 (Las Vegas Algorithm). An algorithm A for a problem class Π is a (generalized) Las Vegas algorithm if and only if it has the following properties:

1. If for a given problem instance π ∈ Π, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.
2. For any given instance π ∈ Π, the run-time of A applied to π is a random variable.
Heavy Tailed Distribution

• In the value distribution of a random variable, some (a few) values can be very big
  – distribution is not bounded
• [Gomes 2000]: backtrack search in combinatorial problems exhibit Heavy-Tailed distribution
  – e.g. 80% runs solve the problem in 1000 iterations but 5% in more than 1,000,000 iterations
• Similar behavior for Local Search
• Thus motivates the use of *restarts*
  – better performance
Heavy Tailed vs. Standard

Power Law Decay

HEAVY TAILED DISTRIBUTION
(infinite mean & variance)

Exponential Decay

Standard Distribution
(finite mean & variance)

[from Gomes 2004]
Histogram of runtime for Magic Square 100x100 with no restart limit
(bins = 1 second)
Zooming on the histogram of runtime for Magic Square 100x100 (4 big runs excluded)
Histogram of runtime for Magic Square 100x100 with restart limit set at 100,000 iterations ($\pm$ 20 seconds)
Statistical Analysis

• Consider the runtime of the algorithm as a random variable $X$
• Use statistical tools to analyze this variable
• Try to find if it matches some well-known probability distribution, e.g.:
  – Exponential distribution
  – Normal (gaussian) distribution
  – Lognormal distribution
• Approximate the runtime by this probability distribution to “predict” runtime behavior
Back to Local Search

• [Hoos / Stutzle 98] proposed to use runtime distributions to characterize running time of LS algorithms for combinatorial optimization

• [Aiex / Resende / Ribeiro 02] showed (experimentally) that the GRASP metaheuristic has exponential runtime distribution on many classical O.R. problems

• [Hoos 97, 98] conjecture that LS solvers for SAT have exponential runtime behavior
Notations

• Let $Y$ be the runtime of a Las Vegas algorithm (or the number of iteration of a LS algorithm)

• Cumulative distribution:

$$F_Y(x) = P[Y \leq x]$$

  - Distribution (probability density function):
    = derivative of cumulative distribution

$$f_Y = F_Y'$$

• Expectation:

$$E[Y] = \int_0^\infty t f_Y(t) dt$$
Basic Exponential Distribution

• Random variable $X \sim \text{Exp}(\lambda)$:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$
(Shifted) Exponential Distribution
Time-To-Target Plots

• For optimization problems, one can consider the runtime of the algorithm to produce a solution of a given target value:
  – Notion of time-to-target
  – Time-To-Target plots

• tttplots system [Aiex / Resende / Ribero 2005]
  – Hypothesis: runtime distributions can be approximated by (shifted) exponential distributions
  – \( \lambda \) and \( x_0 \) computed automatically
    (\( \lambda \) is close to the mean value of runtimes)
tttplots for AS

• As we are considering satisfaction problems, our target value is zero (for objective function)

• Can draw tttplots for runtime execution of AS

• Can check if behavior of runtime of AS algorithm is exponential (or close)

• It depends on the problem!
  – e.g. for N-queens runtimes are (approx.) constant ...
Magic square
200x200
restart limit
set at mean
(500 000 iter.)

$x_0 = -20.57$
$\lambda = 353.85$
Q-Q Plots

• Quantile-Quantile plots (Q-Q plots) can be used to better see how an experimental distribution matches a theoretical one

• Quantile = point below which a certain fraction of distribution lies
  – E.g. the 0.3 quantile is the point at which 30% falls below and 70% above

• Q-Q plot points : (theo. quantile, exp. quantile)

• points on the diagonal = perfect match

• Example: runtime versus exponential distribution
Q-Q plot for Magic Square 100x100 with no restart limit
Q-Q plot for Magic Square 100x100 with restart limit at mean

ms100-M

measured times

exponential quantiles

empirical estimated

0 0.5 1 1.5 2 2.5 3 3.5 4 4.5

0 10 20 30 40 50 60 70
Q-Q plot for Magic Square 100x100 with restart limit at $mean/2$
Parallel Local Search
Parallel Local Search

• Interest since the 90’s in using parallel machines for Local Search

• [Aarts / Verhoeven 95] proposed two simple ways to exploit parallelism in LS:
  – single-walk parallelism
    exploit // within a single search trajectory
  – multi-walk parallelism
    exploit // with different search trajectories
  (= multi-start, = portfolio algorithms for SAT solving)
Parallel Local Search: Single-Walk Parallelism
Parallel Local Search: Multi-Walk Parallelism
Motivation

• Local-Search methods are efficient for Combinatorial Optimization Problems
• Can solve bigger problems with more computing power ...
  e.g. multi-cores CPUs & massively parallel computers

• But can // be exploited efficiently ?
• What about massively parallel computers ?
  e.g. > 10 000 processors
Massively Parallel Machines

K Computer at RIKEN, Kobe
700000 cores
(number 1 at the top 500 in 2011)

Jugene: IBM Blue Gene/P
at Jurich, Germany
300000 cores
(EU PRACE machine)
... at University of Tokyo

Fujitsu Oakleaf FX10
77000 cores

Hitachi HA8000
15000 cores
Top500 Supercomputer Performance Development
From Supercomputer to Desktop...

Intel MIC architecture       NVIDIA Tesla Kepler GPU

50 cores                      2496 cores

ASCI Red: 1 TF
1997 First System 1 TF Sustained
9298 Pentium II Xeon
OS: Cougar
72 Cabinets

Knights Corner: 1 TF
2011 First Chip 1 TF Sustained
1 22nm Chip
OS: Linux
1 PCI express slot
Issues with Parallelism ...

• Good method/algorithm ≠ good performance

• Very different to develop a good method for
  – a few cores (shared memory)
  – a few tens/hundreds of cores (e.g. Grid platforms)
  – a few thousands of cores (e.g. supercomputers)

• The best sequential algorithm may not give the best parallel performances
Existing Parallel Implementations

- Several implementations for shared memory multi-core architectures
  - Local search, SAT, CSP

- A few implementations on PC clusters

- Now some parallel implementations for GPU
  - Genetic algorithms, Tabu Search, Pattern Search

- A few implementation on Grids
  - e.g. Branch & Bound: Grid’BnB
Parallel Implementation of Adaptive Search

• On HA8000 machine at University of Tokyo

• Parallel Method: *independent multi-walks*
  – start // search from different initial configurations
  – no communication

• C-code encapsulated using MPI

• No optimization done wrt architecture (nodes)

• Same code runs on Grid5000 platform
System Overview

The whole nodes (952) are divided into 512, 128, 256, and 56 nodes as sub clusters. The sub clusters are connected with a network.

<table>
<thead>
<tr>
<th>Type</th>
<th>The Number of Nodes</th>
<th>The Number of CPUs</th>
<th>Theoretical Peak (TFLOPS)</th>
<th>Main Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>512</td>
<td>8192</td>
<td>75.366</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2048</td>
<td>18.841</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>4096</td>
<td>37.683</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>36</td>
<td>576</td>
<td>5.299</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>256</td>
<td>2.355</td>
<td>2</td>
</tr>
</tbody>
</table>

- Type A: 512 Nodes
  - Processor: 10 CPUs
  - Memory: 32GB per node
  - Inter-node: 5GB/s

- Type B: 256 Nodes
  - Processor: 10 CPUs
  - Memory: 32GB per node
  - Inter-node: 5GB/s

Storage System: File Server x 16, RAID x 8, Hitachi Stripping File System

- Type A-512 Nodes
- Type A-128 Nodes
- Type B-256 Nodes
- Type B-56 Nodes
Results

![Graph showing speedup vs number of processors for different platforms and problems.]

<table>
<thead>
<tr>
<th>Platform</th>
<th>Problem</th>
<th>Time on 1 core</th>
<th>Speedup on k cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>HA8000</td>
<td>MS 400</td>
<td>6282</td>
<td>10.6</td>
</tr>
<tr>
<td></td>
<td>Perfect 5</td>
<td>42.7</td>
<td>15.0</td>
</tr>
<tr>
<td></td>
<td>A-I 700</td>
<td>638</td>
<td>8.19</td>
</tr>
</tbody>
</table>
Influence of Problem Size:

Magic Squares

<table>
<thead>
<tr>
<th># cores</th>
<th>MS 100</th>
<th></th>
<th>MS 120</th>
<th></th>
<th>MS 200</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>speed</td>
<td>time</td>
<td>speed</td>
<td>time</td>
<td>speed</td>
</tr>
<tr>
<td>1</td>
<td>18.2</td>
<td>1.0</td>
<td>53.4</td>
<td>1.0</td>
<td>338</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>2.16</td>
<td>8.41</td>
<td>5.84</td>
<td>9.14</td>
<td>42.3</td>
<td>8.0</td>
</tr>
<tr>
<td>16</td>
<td>1.69</td>
<td>10.8</td>
<td>3.99</td>
<td>13.4</td>
<td>22.4</td>
<td>15.1</td>
</tr>
<tr>
<td>32</td>
<td>1.43</td>
<td>12.7</td>
<td>3.03</td>
<td>17.7</td>
<td>14.8</td>
<td>22.9</td>
</tr>
<tr>
<td>64</td>
<td>1.20</td>
<td>15.1</td>
<td>2.26</td>
<td>23.6</td>
<td>12.2</td>
<td>27.8</td>
</tr>
<tr>
<td>128</td>
<td>1.16</td>
<td>15.5</td>
<td>2.24</td>
<td>23.9</td>
<td>12.1</td>
<td>28.0</td>
</tr>
</tbody>
</table>
Costas Array Problem

• Proposed by Costas in the 60’s
• Sonar / Radio applications
• Active community for 40 years
• Constructive methods exist
  – but not valid for all $n$
• Very combinatorial, solutions are very scarce
  – $n=29$ has only 164 solutions (23 unique) out of 29!
• still several open problems ...
  – Does it exist costas arrays for $n=32$ ?
Linear speedups for large instances

Speedups for CAP 22
Costas Array, up to 8192 cores
Current Research

• Develop new models for // Local Search

• Communication between search engines
  – dependent multi-walks

• What to communicate / when to communicate?

• Idea of a pool of (elite) solutions
  – ...