

**CNGL**  
Centre for Next Generation Localisation

**Optimising Parameters with Dynamic Task-Farming**  
Joachim Wagner 2011-05-18

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

**CNGL**  
Centre for Next Generation Localisation

**Outline**

- Setting: optimising parameters
- Hill climbing
- Parallelisation of hill climbing
- Dynamic task-farming
- Over-fitting
- Noise

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

**CNGL**  
Centre for Next Generation Localisation

**Training Setup**

- Training set, dev set 1, dev set 2, test set
- Splitting dev sets further. e.g. for sampling results obtained with different random seeds
- Issues observed by Anton
- Objective Function
- Are we training for the right thing for all possible applications?

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

**CNGL**  
Centre for Next Generation Localisation

**Hill Climbing**

- Move towards higher ground (up the hill)
- Usually described for discrete search graphs

Frances Linzee Gordon, allposters.co.uk #3644263

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

**CNGL**  
Centre for Next Generation Localisation

**Example for Discrete Graph**

- 4 binary features -> 16 possible configurations

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

**CNGL**  
Centre for Next Generation Localisation

**Example for Discrete Graph (2)**

- Re-define neighbourhood: also remove features

VNDP SFI DCU University College Dublin University of Limerick Trinity College Dublin

### Simulated Annealing

- Physics: seeking state of lowest energy
- Temperature allows search to explore states at higher energy levels
- Annealing: reduce temperature slowly

Energy

better

### Quantum Annealing

- Physics: seeking state of lowest energy
- Quantum tunneling on short distance

Energy

better

Too far

### Quantum Tunnel in Discrete Graph

- Left to right, tunnel across one steps

### Continuous Case

- Example: optimising feature weights
- Grid method
  - Dimensionality -> many neighbours
- Sampling within search radius / sphere
  - One sample per time step
  - Move (climb) to new point if better
  - Blind to barriers, implicit quantum tunnelling
  - Metric (shape of search sphere)
  - Dimensionality
  - Computational challenge of *uniform* sampling
  - Search scale decay, implicit quantum annealing
  - Simulated annealing: randomly allow steps down (within limits)

### Comparing Search Strategies

- Unexpected results

		# steps						
		0	50	100	200	400	800	
Decay	IScale	0	50	100	200	400	800	
	exp	0.5	52.92%	56.87%	59.07%	59.67%	59.73%	59.73%
	exp	1	58.35%	59.94%	60.15%	60.26%	60.34%	60.34%
	exp	2	61.37%	60.05%	60.05%	60.60%	60.69%	60.71%
	exp	4	57.26%	60.22%	60.42%	60.67%	60.97%	61.01%
exp	8	57.62%	59.46%	60.57%	60.85%	60.89%	60.93%	
linear	IScale	0	50	100	200	400	800	
	linear	0.5	58.11%	59.04%	60.03%	60.30%	60.43%	60.45%
	linear	1	55.83%	58.93%	59.86%	60.19%	60.27%	60.29%
	linear	2	54.67%	58.69%	59.40%	60.37%	60.69%	61.04%
	linear	4	57.35%	60.25%	60.34%	60.36%	60.39%	60.39%
linear	8	55.72%	58.67%	59.48%	60.42%	60.60%	60.76%	
none	IScale	0	50	100	200	400	800	
	none	5	55.88%	60.46%	60.70%	60.95%	61.03%	61.20%
	none	5	54.77%	59.28%	60.31%	61.01%	61.10%	61.35%
	none	5	57.94%	60.27%	60.78%	61.34%	61.56%	61.56%
	none	5	58.18%	60.44%	61.53%	61.58%	61.60%	61.91%
none	5	57.02%	59.96%	60.61%	61.53%	62.13%	62.17%	

### Tuning

- Cookbook approach may not work well -> you have to tune your search
- Logarithmic scale for weights, why?
- Limit range to reasonable values (can be wrong)

**CNGL**  
Center for Next Generation Localisation

### Options for Parallelisation of Search

- Easy:
  - Training size, language pairs etc. 20
  - Cross-validation runs x 10
  - Shots (different starting points) x 10
  - Trying different objective functions x 3 = 6,000
- More programming: split devset and process test items in parallel
- Change search: explore multiple neighbours at once before making a decision where to move to
  - Waiting for fixed number of neighbours?
  - Idle time of machine

INSP DCU

**CNGL**  
Center for Next Generation Localisation

### Presentational Considerations

- Dependency on speed on machines, network state etc. vs deterministic search
- Ease of mathematical description

$$b_{i,j} = \begin{cases} x_{i,j-1} & \text{if } f_{\text{obj}}(x_{i,j-1}) > f_{\text{obj}}(b_{i,j-1}), \\ b_{i,j-1} & \text{otherwise} \end{cases}$$

INSP DCU

**CNGL**  
Center for Next Generation Localisation

### Example: Parallel Exploration of Neighbours

- 12 neighbours, 3 CPUs

CPU 1	1	4	7	12	← idle
CPU 2	2	6	9	10	012
CPU 3	3	5	8	11	← idle

time →

INSP DCU

**CNGL**  
Center for Next Generation Localisation

### Cluster Hardware

192 cores  
12 cores  
768 cores

INSP DCU

**CNGL**  
Center for Next Generation Localisation

### Static Task-Farming

PBS Job Description → Taskfarming Executable (n instances)

Taskfarming Executable (n instances) → 1 Master ↔ n-1 Worker

Task file (.tfm): one task per line → 1 Master

1 Master ↔ n-1 Worker (MPI or HTTP Communication)

n-1 Worker → Task execution (child process)

INSP DCU

**CNGL**  
Center for Next Generation Localisation

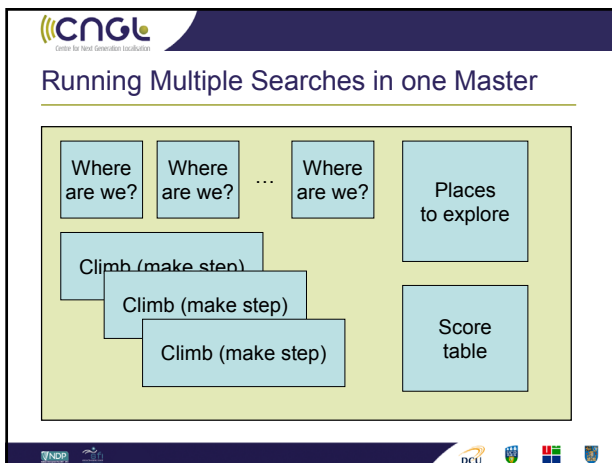
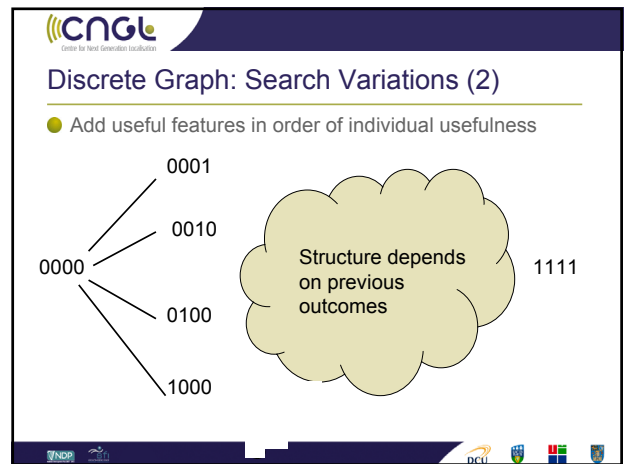
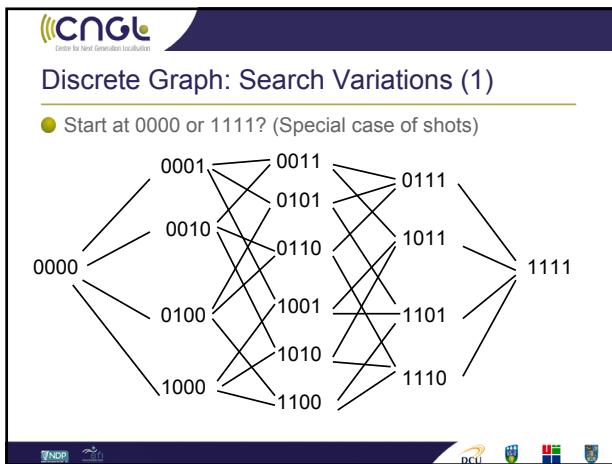
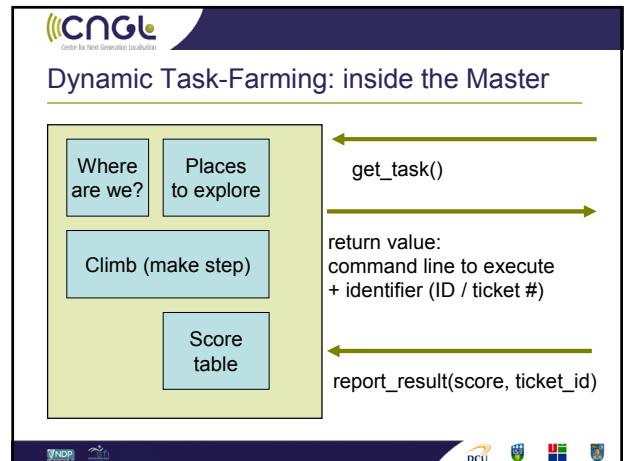
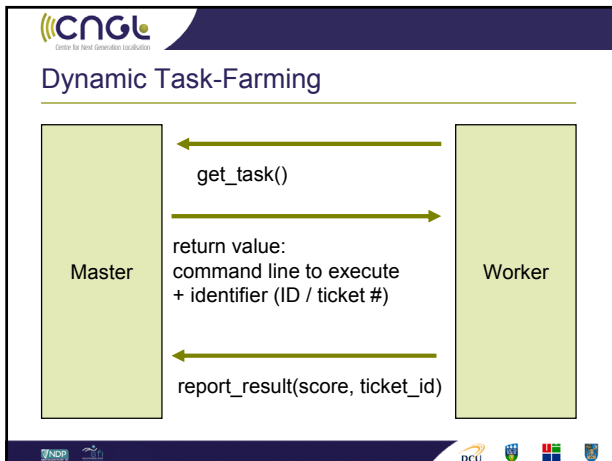
### Static Task-Farming (2)

Master ↔ Worker

Worker → Master: get\_task()

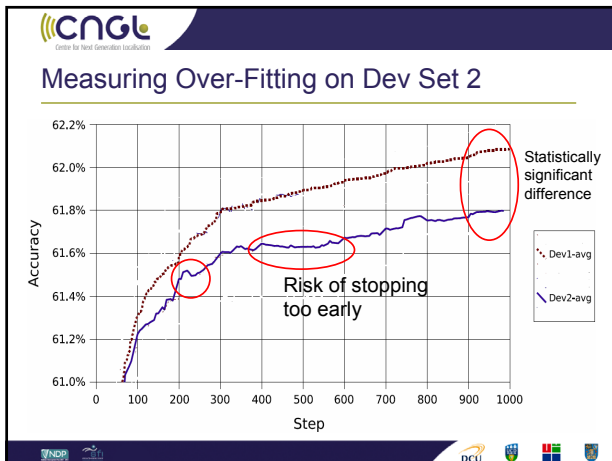
Master → Worker: return value: command line to execute

INSP DCU

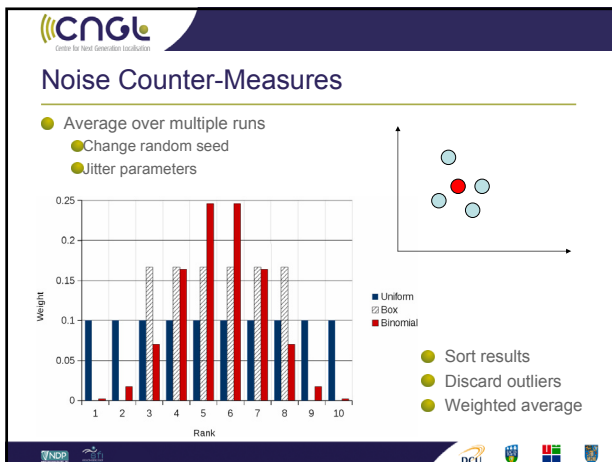


**When to Stop Searching: Over-Fitting**

- Hill climbing decisions based on dev set 1 results
- Use dev set 2 to detect over-fitting  
"learner improves its predictions on training items at the price of making worse decisions for unseen test items"
- By construction of hill climbing (without annealing), we always improve on dev set 1  
-> overfitting == deterioration on dev set 2



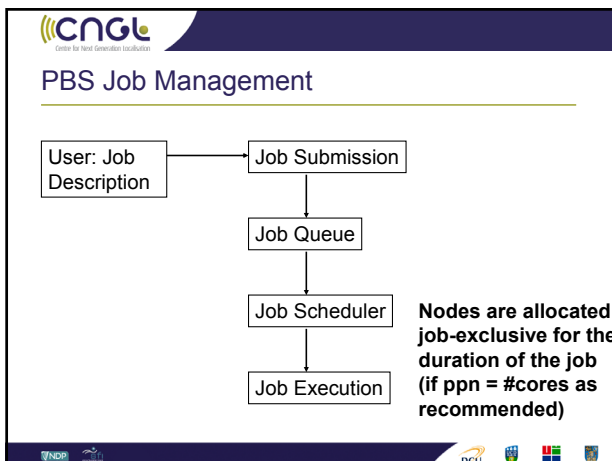
- ### Noisy Dev Set Results
- Examples:
    - Random seed affects results: Lorg, MERT
    - Small changes of parameters may have random effects on results: unstable learning methods (e.g. decision trees)
    - Hill climbing step may be sub-optimal
    - Noise creates local optima
- 



### Questions

?

Contact:  
 Joachim Wagner  
 CNGL System Administrator  
 jwagner@computing.dcu.ie  
 (01) 700 6915



- ### PBS Job Management Commands
- `qsub myjob.pbs`
    - submits a job
    - PBS description: shell script with #PBS commands (ignored by shell, see next slide)
  - `qstat, qstat -f jobnumber`
  - `qdel jobnumber`
  - `pbsnodes -a`
    - list all nodes with status and properties

**CNGL**  
Center for Next Generation Localization

## PBS Job Description

```
#!/bin/sh

# Common PBS variables
#PBS -l nodes=5:ppn=8:mem16GB
#PBS -N test-mpi
#PBS -M jwagner@computing.dcu.ie
#PBS -m eba
#PBS -l walltime=00:05:00

source ${HOME}/.bashrc

exp_dir=/home/jwagner/
cd ${exp_dir}

# run multiple copies
mpiexec -n 40 /home/jwagner/intro/demo.py
```

**Number of nodes**  
**#CPU cores/node**

**Notification: end, begin and abort**

**Maximum runtime**

**Number of processes to start**

**CNGL**  
Center for Next Generation Localization

## Example: Memory-Intensive Job

```
#!/bin/sh

# Common PBS variables
#PBS -l nodes=1:ppn=8:min32GB
#PBS -N my-test-job
#PBS -M jwagner@computing.dcu.ie
#PBS -m eba
#PBS -l walltime=01:30:00

source ${HOME}/.bashrc

exp_dir=/home/jwagner/
cd ${exp_dir}

# run single process
/home/jwagner/intro/hello.py
```

**CNGL**  
Center for Next Generation Localization

## Taskfarming Executable

- If Instance ID == 0
  - Run master code loop:
    - Read .tfm file (arg 1)
    - Send lines to worker
    - Exit if no more task and all worker finished
- Else
  - Run worker loop:
    - Ask master for a task
    - Execute task
    - Exit if master has no more tasks

**CNGL**  
Center for Next Generation Localization

## Taskfarming Options

- Using individual PBS jobs
  - Can only allocate resources in multiples of 1/8 or 1/4
    - Example: 3 GB task -> 4 GB job (ppn=2:cores8:mem16GB)
  - Floods job queue
- MPI-based taskfarming
  - All tasks inside one job
    - Example: 3 GB task -> 5 workers per 16 GB node
  - Master blocks one CPU core
- HTTP/XML-RPC-based taskfarming
  - Master runs on maia login node
  - Workers can run in multiple jobs
    - Example: 3 GB tasks -> one job with 5 workers for 16 GB nodes and one job with 8 workers for 32 GB nodes

**CNGL**  
Center for Next Generation Localization

## Example: Taskfarming PBS File

```
#!/bin/bash

# Common PBS variables
#PBS -l nodes=2:ppn=4:min4GB
#PBS -N testxle5
#PBS -M jwagner@computing.dcu.ie
#PBS -m bea
#PBS -l walltime=01:30:00
#PBS -V

source ${HOME}/.bashrc

exp_dir=/home/jwagner/xle_parsing/
cd ${exp_dir}

mpiexec -n 8 /home/jwagner/taskfarm.py \
/home/jwagner/xle_parsing/xle.tfm
```

**CNGL**  
Center for Next Generation Localization

## Example: Taskfarming TFM File

```
/home/jwagner/xle_parsing/run-package.sh 000
/home/jwagner/xle_parsing/run-package.sh 001
/home/jwagner/xle_parsing/run-package.sh 002
/home/jwagner/xle_parsing/run-package.sh 003
/home/jwagner/xle_parsing/run-package.sh 004
/home/jwagner/xle_parsing/run-package.sh 005
/home/jwagner/xle_parsing/run-package.sh 006
/home/jwagner/xle_parsing/run-package.sh 007
/home/jwagner/xle_parsing/run-package.sh 008
/home/jwagner/xle_parsing/run-package.sh 009
/home/jwagner/xle_parsing/run-package.sh 010
/home/jwagner/xle_parsing/run-package.sh 011
/home/jwagner/xle_parsing/run-package.sh 012
```

**Example: Taskfarming Helper Script**

```
#!/bin/bash
PACKAGE=$1
# never use more than 995 MB
ulimit -v 995000
# keep time
touch /home/jwagner/xle_parsing/Files/$PACKAGE.start
# prepare data
bunzip2 /home/jwagner/xle_parsing/Files/$PACKAGE.bz2
# run parser
/usr/local/xle/bin/xle -noTk -e "create-parser /usr/local/share/xle-grammars/english/english.lfg; parse-testfile /home/jwagner/xle_parsing/Files/$PACKAGE; exit" 2>/dev/null >/dev/null
# delete files not needed anymore
rm -f /home/jwagner/xle_parsing/Files/$PACKAGE
rm -f /home/jwagner/xle_parsing/Files/$PACKAGE.new
# compress results
bzip2 /home/jwagner/xle_parsing/Files/$PACKAGE.stats
```

**run-package.sh**

**Example: Non-Terminating Task**

CPU 1: 000, 003, 006 (does not terminate)

CPU 2: 001, 005, 008, 009, 011 → idle

CPU 3: 002, 004, 007, 010, 012 → idle

CPU 4: Master: reads .tfm and distributes tasks

**Killed at  
Walltime  
Limit**

**Estimating the PBS Walltime Parameter**

- Collect durations from test run
- Usually high variance of execution time
  - Long sentences
  - Parameters
- Don't use #packages x avg. time per package
  - High risk (~50 %) that more time is needed
  - Prefix jobname with, for example, "24h-"
- Random sampling with observed package durations: /home/jwagner/tools/walltime.py

**Effect of Task Size**

- Job will wait for last task to finish (or be killed when walltime limit is reached)
- What if a task crashes?
  - Results are incomplete
  - Next tasks is executed
- What if a task does not terminate?
  - Results are incomplete
  - Fewer CPUs available for remaining tasks
- Overhead of starting tasks