Evaluating Players’ *HOMM-III* Skill-Selection Strategies in Parallel

Dimitris Diochnos

University of Illinois at Chicago
Dept. of Mathematics, Statistics, and Computer Science

April 23, 2008
Outline

1. Introduction
2. Some skill-selection mechanisms
3. Justifying the title
4. On the implementation
5. Experiments and Results
6. Future Work
1. Introduction
   - A general framework of the problem
   - The real deal

2. Some skill-selection mechanisms

3. Justifying the title

4. On the implementation

5. Experiments and Results

6. Future Work
Some objects

- A group of resources $\mathcal{R}$, such that $|\mathcal{R}| = N$
- A basket $\mathcal{B}$ with $M$ slots; $M < N$.
- Each slot is composed by $k$ pockets.
  - Pockets in the same slot contain *same* resources.
  - Pockets in different slots contain *different* resources.
Assigning resources to pockets

- At each step we are offered 2 choices of resources.
  - One resource does not appear in *any* slot so far.
  - The other resource appears in a *non-empty* slot.
- We pick one of them and assign it to a pocket (appropriate slot).
- The process is repeated until all pockets are *non-empty*. 
Computational Problem

\[ \Pr[\mathcal{R}_i \in \mathcal{B}] \]

The probability depends on the way we select resources
The adversary

Natural questions:

- How are the resources offered?
- What is the probability that $\langle R_i, R_j \rangle$ is an offer?

There is a weight function $w : R \rightarrow \mathbb{N}$, such that $\sum_{i=1}^{N} w(R_i) = c$

The current setting forms a partition on resources.

- $\Pr[R_i] = w(R_i) / \sum_{\alpha : R_\alpha \notin B} w(R_\alpha)$ \hspace{1cm} $[R_i \notin B]$

- Consider $S = \bigcup s_j$, such that for each $s_j : 1 < \text{used pockets} < k$

  $\Pr[R_j] = w(R_j) / \sum_{\beta : R_\beta \in S} w(R_\beta)$ \hspace{1cm} $[R_j \in S]$
The adversary

Natural questions:

- How are the resources offered?
- What is the probability that $\langle R_i, R_j \rangle$ is an offer?

There is a weight function $w : R \rightarrow \mathbb{N}$, such that $\sum_{i=1}^{N} w(R_i) = c$

The current setting forms a partition on resources.

- $\Pr[R_i] = \frac{w(R_i)}{\sum_{\alpha : R_\alpha \not\in B} w(R_\alpha)}$ $[R_i \not\in B]$
- Consider $S = \bigcup s_j$, such that for each $s_j : 1 < \text{used pockets} < k$. $\Pr[R_j] = \frac{w(R_j)}{\sum_{\beta : R_\beta \in S} w(R_\beta)}$ $[R_j \in S]$
The adversary

Natural questions:

- How are the resources offered?
- What is the probability that $\langle R_i, R_j \rangle$ is an offer?

There is a weight function $w : R \rightarrow \mathbb{N}$, such that $\sum_{i=1}^{N} w(R_i) = c$

The current setting forms a partition on resources.

- $\Pr[R_i] = \frac{w(R_i)}{\sum_{\alpha : R_\alpha \notin B} w(R_\alpha)}$ \hspace{1cm} [$R_i \notin B$]

- Consider $S = \cup s_j$, such that for each $s_j : 1 <$ used pockets $< k$.
  \hspace{1cm} $\Pr[R_j] = \frac{w(R_j)}{\sum_{\beta : R_\beta \in S} w(R_\beta)}$ \hspace{1cm} [$R_j \in S$]
The adversary

Natural questions:

- How are the resources offered?
- What is the probability that $\langle R_i, R_j \rangle$ is an offer?

There is a weight function $w : \mathcal{R} \rightarrow \mathbb{N}$, such that $\sum_{i=1}^{N} w(R_i) = c$

The current setting forms a partition on resources.

- $\Pr[R_i] = \frac{w(R_i)}{\sum_{\alpha : R_\alpha \notin B} w(R_\alpha)}$ \quad [R_i \notin B]
- Consider $S = \bigcup s_j$, such that for each $s_j : 1 < \text{used pockets} < k$.
  $\Pr[R_j] = \frac{w(R_j)}{\sum_{\beta : R_\beta \in S} w(R_\beta)}$ \quad [R_j \in S]
More constraints ...

- Is this all? **NO!**
- There is also a *model* that influences offers \( \langle R_i, R_j \rangle \).
  - The resources are partitioned into groups.
  - Each group \( g \) is associated with a period \( p_g \).
  - Every \( p_g \) timesteps a resource belonging to \( g \) has to appear.
  - Formulas for computing probabilities change.
More constraints ...

- Is this all? NO!
- There is also a *model* that influences offers $\langle R_i, R_j \rangle$.
  - The resources are partitioned into groups.
  - Each group $g$ is associated with a period $p_g$.
  - Every $p_g$ timesteps a resource belonging to $g$ has to appear.
  - Formulas for computing probabilities change.
Is this all?  NO!

There is also a *model* that influences offers $\langle R_i, R_j \rangle$.

- The resources are partitioned into groups.
- Each group $g$ is associated with a period $p_g$.
- Every $p_g$ timesteps a resource belonging to $g$ has to appear.
- Formulas for computing probabilities change.
Motivation

HOMM-III: Heroes of Might and Magic III by New World Computing

- Turn based strategy game
- Released: June 1999
- 3 different world-wide tourneys
- Many country-level tourneys
- Increased popularity:
  - Russia
  - Germany
  - Poland
  - Bulgaria
A brief description

- Users control heroes that acquire different skills \( (M = 8) \)
- The resources reflect different skills \( (N = 28) \) (e.g. Tactics, Wisdom, Earth Magic)
- Resources in the basket reflect skills acquired by a hero
- Each skill has \( k = 3 \) different levels of expertize (Basic, Advanced, Expert)

Groups of skills:

- **WISDOM**: Wisdom \( [\rho = 6] \)
- **MAGIC**: Air, Earth, Fire, and Water Magic \( [\rho = 4] \)
- **REST**: All other 23 skills \( [\rho = \infty] \)

Offer / level: \( \langle \) Upgrade an existing skill, Get a new skill \( \rangle \)

Computational Problem: What is the probability that a specific hero obtains a specific skill given a skill-selection mechanism?
A brief description

- Users control heroes that acquire different skills \((M = 8)\)
- The resources reflect different skills \((N = 28)\)
  (e.g. Tactics, Wisdom, Earth Magic)
- Resources in the basket reflect skills acquired by a hero
- Each skill has \(k = 3\) different levels of expertize
  (Basic, Advanced, Expert)
- Groups of skills:
  - WISDOM: Wisdom \([\rho = 6]\)
  - MAGIC: Air, Earth, Fire, and Water Magic \([\rho = 4]\)
  - REST: All other 23 skills \([\rho = \infty]\)
- Offer / level: \(\langle\text{Upgrade an existing skill, Get a new skill}\rangle\)

Computational Problem: What is the probability that a specific hero obtains a specific skill given a skill-selection mechanism?
A sample hero
**Always Right (AR)**

Recall: Offer / level: \(\langle\) Upgrade an existing skill, Get a new skill \(\rangle\)

\[AR\ does\ not\ depend\ on\ the\ user’s\ preference\ of\ skills\]

**AR**: Small state-space \(\implies\) Brute-force computation

\(\text{(double precision suffices)}\)

**Note**: \(AR\) can be used to verify the model since we have no source code!

Some skill-selection mechanisms

Other skill-selection mechanisms

**AL:** Always pick the left offer.

**ALTP:** "Always Left Then Preference"
Upgrade existing skills as long as *not* all of them at Expert level. Then the offer will be: ⟨ New Skill A, New Skill B ⟩

**SPOU:** "Seek Preference Otherwise Upgrade"
If an interesting (new) skill is offered, pick it; otherwise upgrade an existing skill.

Curse of dimensionality! $\implies$ Monte Carlo.
Other skill-selection mechanisms

**AL**: Always pick the left offer.

**ALTP**: "Always Left Then Preference"
Upgrade existing skills as long as *not* all of them at Expert level. Then the offer will be: \( \langle \text{New Skill A, New Skill B} \rangle \)

**SPOU**: "Seek Preference Otherwise Upgrade"
If an interesting (new) skill is offered, pick it; otherwise upgrade an existing skill.

- Curse of dimensionality! \( \implies \) Monte Carlo.
Outline

1. Introduction
2. Some skill-selection mechanisms
3. Justifying the title
4. On the implementation
5. Experiments and Results
6. Future Work
Why `internals_mc`?

`internals`: Solvers are hosted on the thread:

*On the internals of offered skills when leveling-up a hero*


`mc`: Monte Carlo

`internals_mc` is the solver for skill-selection mechanisms which imply large state-spaces. [page 8 on thread above]
Why multiple cores?

Because if we want precision on the results and good confidence then we need many runs!

- View the probability each skill has under a specific strategy (policy) on accepting skills as a random variable.
- Then independent runs are Bernoulli trials.

Say that we want to be 95% confident that the computed probability of a specific skill is correct with at least $k$ digits (on a specific skill). Then:

**Chebychev bound:** $\#\text{runs} \geq 5 \cdot 10^{2k}$

**Central Limit Theorem:** $\#\text{runs} \geq 0.9604 \cdot 10^{2k}$

- For precision of at least 3 decimal digits we already need about a million runs! [And we want the entire distribution ...]
Outline

1. Introduction
2. Some skill-selection mechanisms
3. Justifying the title
4. On the implementation
   - General
   - Data structures that are passed on threads
   - Sample Run
5. Experiments and Results
6. Future Work
On the implementation

Homepage: HeroesCommunity


Post: internals_mc: Evaluation of user’s policy with Monte Carlo methods

Implemented in C++ with Pthreads

- Current version: 2.0
- 4,529 lines of code

Compiles under any platform

- Pthreads under Windows:
  
  Open Source POSIX Threads for Win32 [R. Johnson]
  
  http://sourceware.org/pthreads-win32/

- gettimeofday() is re-defined and conditionally included on compile time under Windows.
A general scheme

Important: Random numbers should not be correlated!
  - That’s an active area of research on its own

- $N + 1$ threads:
  - $N$ are workers
  - 1 is the generator

- Each worker is associated with a queue which contains sequences of random numbers generated by \texttt{rand()} in successive calls

- As soon as the generator fills a queue with the appropriate amount of data, a signal is sent to the associated thread to start working
A general scheme

Important: Random numbers should not be correlated!
  - That’s an active area of research on its own

- $N + 1$ threads:
  - $N$ are *workers*
  - 1 is the *generator*

- Each *worker* is associated with a *queue* which contains sequences of random numbers generated by `rand()` in successive calls

- As soon as the generator fills a queue with the appropriate amount of data, a signal is sent to the associated thread to start working
Schematically

generator

queue 1

worker 1

queue 2

worker 2

queue 3

worker 3

......

queue N

worker N
Schematically

![Diagram showing a generator connected to multiple queues and workers](image-url)
Schematically

1. Generator
2. Signal
3. Queues
4. Workers
Schematically

And so on ...
A memory concern ...

- Generating a random number is relatively cheap compared to a single simulation run.
- Each run requires $\leq 2 \cdot 2^2 = 44$ random numbers to determine the offers / level.
- We need $44 \cdot 4 \cdot |R| = 176|R|$ bytes in the worst case! ($|R|$ is a multiple of a million ...)
- We can not truncate random sequences and use a single byte; although in all mod operations we will never use a number greater than 112 !!
  
  e.g. $(4 \mod 3) \mod 2 \neq 4 \mod 2$
Generator thread

typedef struct {
    pthread_mutex_t **queue_mutex_array;
    pthread_cond_t **queue_cond_array;
    QUEUE_PTR *queue_array;
    bool **waiting_array;

    int id;

    int items_per_chunk;
} GENERATOR_DATA;
typedef struct {
    pthread_mutex_t * queue_mutex;
    pthread_cond_t * queue_cond;
    QUEUE_PTR queue;
    bool * waiting;

    int id;

    pthread_mutex_t * total_episodes_mutex;
    long unsigned int * total_episodes_atm;
    long unsigned int total_episodes_computed_by_threads;

    long unsigned int episodes_to_compute;

    MONTE_CARLO_PTR myMC;
    MT_PTR myMT;
} WORKER_DATA;
On the implementation

Sample Run

```
Terminal — bash — 81x35

gunnar:~/internalsMCthreads$ ./internals_mc demonic marius al 1000000
MARIUS
--------
AIR MAGIC    =---------------------------=>  45.70 %
ARCHERY      ===============> 26.56 %
ARMORER      ==================================> 100.00 %
ARTILLERY    ===============> 22.53 %
BALLISTICS   ===============> 30.26 %
DIPLOMACY    ===============> 10.52 %
EAGLE EYE    ===============> 14.19 %
EARTH MAGIC  ===============> 57.75 %
ESTATES      ===============> 14.19 %
FIRE MAGIC   ===============> 65.72 %
FIRST AID    ===============>  9.64 %
INTELLIGENCE ===============>  9.64 %
LEADERSHIP   ===============> 14.18 %
LEARNING     ===============> 18.52 %
LOGISTICS    ===============> 40.40 %
LUCK         ===============>  9.64 %
MYSTICISM    ===============>  9.64 %
NAVIGATION   ===============> 18.52 %
NECROMANCY   = 0.00 %
OFFENSE      ===============> 33.92 %
PATHFINDING  ===============> 18.52 %
RESISTANCE   ===============> 26.56 %
SCHOLAR      ===============>  9.64 %
SCOUTING     ===============> 22.63 %
SORCERY      ===============> 14.18 %
TACTICS      ===============> 26.56 %
WATER MAGIC  ===============> 26.22 %
WISDOM       ===============> 94.01 %

Duration   : 7.926000 seconds.
Seed        : 1280591625

gunnar:~/internalsMCthreads$
```
Outline

1. Introduction
2. Some skill-selection mechanisms
3. Justifying the title
4. On the implementation
5. Experiments and Results
6. Future Work
Testbed machine

- 2.2 GHz Intel Core 2 Duo processor
- 2 GB of RAM
- 4 MB of L2 Cache
- gcc version 4.0.1 (Apple Inc. build 5465)
- Mac OS X 10.5.2
## Experiments and Results

### Running times

<table>
<thead>
<tr>
<th>Hero</th>
<th>Serial</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AR</td>
<td>AL</td>
</tr>
<tr>
<td>Thane</td>
<td>65.76</td>
<td>71.00</td>
</tr>
<tr>
<td>Crag Hack</td>
<td>64.69</td>
<td>71.70</td>
</tr>
<tr>
<td>Rashka</td>
<td>65.53</td>
<td>71.86</td>
</tr>
<tr>
<td>Orrin</td>
<td>63.87</td>
<td>70.92</td>
</tr>
<tr>
<td>Ivor</td>
<td>63.89</td>
<td>71.24</td>
</tr>
</tbody>
</table>

**Table:** Running times (secs) for simulating 5 million episodes.
## Speedup

<table>
<thead>
<tr>
<th>Hero</th>
<th>Speedup</th>
<th>AR</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thane</td>
<td>1.76</td>
<td>1.82</td>
<td></td>
</tr>
<tr>
<td>Crag Hack</td>
<td>1.73</td>
<td>1.71</td>
<td></td>
</tr>
<tr>
<td>Rashka</td>
<td>1.77</td>
<td>1.86</td>
<td></td>
</tr>
<tr>
<td>Orrin</td>
<td>1.73</td>
<td>1.82</td>
<td></td>
</tr>
<tr>
<td>Ivor</td>
<td>1.72</td>
<td>1.83</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Speedup achieved.
## Efficiency

<table>
<thead>
<tr>
<th>Hero</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$AR$</td>
</tr>
<tr>
<td>Thane</td>
<td>0.88</td>
</tr>
<tr>
<td>Crag Hack</td>
<td>0.87</td>
</tr>
<tr>
<td>Rashka</td>
<td>0.89</td>
</tr>
<tr>
<td>Orrin</td>
<td>0.87</td>
</tr>
<tr>
<td>Ivor</td>
<td>0.86</td>
</tr>
</tbody>
</table>

**Table:** Efficiency achieved.
Experiments and Results

Speedup and Efficiency are actually better ...

The previous strategies on picking skills are not sophisticated!

Under *ALTP* or *SPOU* policies heroes can achieve:

- Speedup 1.93
- Efficiency 0.97
Outline

1. Introduction
2. Some skill-selection mechanisms
3. Justifying the title
4. On the implementation
5. Experiments and Results
6. Future Work
What’s left for a future release?

- Eliminate the use of a generator thread and directly assign a random number generator to each thread. Available options:
  - Parallel Random Number Generation, [S. Skiena, Stony Brook University]
  - The Scalable Parallel Random Number Generators Library (SPRNG), [M. Mascagni, H. Chi, and J. Ren, Florida State University]

Considerations:

  +: Trivial requirements on RAM
  -: Is performance comparable to \texttt{rand()}?

- Allow results for intermediate levels as well, since they are of practical importance
- Implement other selection-strategies as well
Questions ?