Allis, L.V., Herik, H.J. van den, and Huntjens, M.P.H. (1996). Go-Moku Solved by New Search Techniques. *Computational Intelligence: An International Journal*, Vol. 12, No. 1, pp. 7-24. Special Issue on Games. Blackwell Publishers. ISSN 0824-7935.

Allis, L.V., Meulen, M. van der, and Herik, H.J. van den (1994). Proof-Number Search. *Artificial Intelligence*, Vol. 66, No. 1, pp. 91-124. ISSN 0004-3702.

Sakata, G. and Ikawa, W. (1981). *Five-In-A-Row, Renju*. The Ishi Press, Inc. Tokyo, Japan.

Patashnik, O. (1980). Qubic: 4x4x4 Tic-Tac-Toe, *Mathematics Magazine*, Vol. 53, pp. 202-216.

REVIEW

MULTIGAME — AN ENVIRONMENT FOR DISTRIBUTED GAME-TREE RESEARCH

Ph.D. thesis
by John W. Romein
Vrije Universiteit Amsterdam, 176 pp.
Postscript version available from: http://www.cs.vu.nl/~john/thesis

Reviewed by Dap Hartmann[1]

In the spirit of the new, broader scope of the ICGA, John Romein's Ph.D. thesis describes an environment in which a wide variety of one- and two-person games can be researched on parallel computing platforms. By creating a very high-level language, he supplies computer game researchers with a computing environment in which search experiments on distributed game trees can be carried out without the burden of having to deal with the intricacies of parallelism. The rules of most games can be readily described in this Multigame language. Such a program is then translated by a front-end compiler into a move generator for that game. Together with a user-supplied evaluation function in C code, the move generator is linked to the Multigame runtime system code, and compiled into a game-playing program. With this clever approach, parallel programming issues such as communication, synchronization, work and data distribution, and deadlock prevention are hidden from the programmer. This will undoubtedly come as quite a relief to many researchers. However, there is a performance price to pay, as the C code from the front-end compiler can be an order of magnitude slower than a hand-crafted move generator. But the user may choose to provide such C code himself, and still benefit from the pre-fab parallelism that the platform provides.

Multigame applies to one- and two-person board games with perfect information. 'Board games' is a flexible concept which may also include one-person games that can be mapped onto a board representation. An example is Rubik's Cube, where the six sides of the cube are projected on a board consisting of 54 squares, each representing one of the visible faces of the puzzle's sub-cubes. Programs written in the Multigame language are surprisingly compact. A move generator for the game of Domineering requires just 15 lines of Multigame code, while the move generator for solving Rubik's Cube can be represented in 462 lines. A complete move generator for the game of chess needs only 208 lines of code. For example, the following fragment generates knight moves:

```
knight_move =
        find own knight,
        pick up,
        orthogonal,
        step,
        either turn 45 or turn –45,
        step,
        not points at own piece,
        put down.
```

[1] Binnendoor 14, 2512 XX Den Haag, The Netherlands. Email: dap_hartmann@yahoo.com.

For solving one-person games (puzzles), the Multigame platform uses the Iterative Deepening A* (IDA*) algorithm developed by Korf. For two-person games, there is a choice between the Alpha-Beta, NegaScout, and MTD(f) algorithms.

After thorough descriptions and discussions of the Multigame language (Chapter 3) and the parallel search engines (Chapter 4), the meat is delivered in Chapter 5. Here, Romein presents the most innovative part of his research: Transposition-table Driven (work) Scheduling (TDS), which he successfully implemented for the IDA* algorithm. "The key idea is to partition the transposition table over the processors and to *migrate* a state to the processor that owns the corresponding table entry, rather than using work stealing." This idea seems counter-intuitive at first, because TDS requires a great deal of data communication. But the big advantage lies in the fact that the transposition table is now always local to the processor working on the transferred state, whereas traditional 'work stealing' approaches must access remote transposition tables. Instead of stealing work from the work queues of other processors, in TDS a processor waits for other processors to send it work, namely jobs on states that have signatures referring to that processor's transposition table. This way, no two processors will ever work on the same sub-tree concurrently. Another unusual aspect of TDS is that the processor which is assigned the work does not report back its results to the processor that supplied the work. Only when such a 'slave without a master' finds a solution at the current search iteration, does it raise a global flag signifying its success. For this reason, TDS must synchronize the processors after each IDA* iteration. In several experiments, Romein convincingly shows that TDS clearly outperforms conventional work-stealing algorithms, typically by a factor of 2 to 3. Tested on systems with 2^N (N = 0…7) processors, TDS achieved a nearly linear speed-up with the number of processors. Running on 128 processors, TDS realized speed-ups between 109 and 122.

The big question left open is whether TDS can also be applied to two-person games, which do require back-propagation of search results. It is a bit disappointing that Romein has little more to say about this important issue than "More research is necessary to evaluate the applicability of TDS to this class of search algorithms." Even though the thesis as a whole is a nice piece of work, it is obvious that TDS is its saving grace, inspired perhaps by that kindred spirit who has motivated so many young computer-science students. Doing original research is hard work, and original ideas do not always present themselves when they are needed most. From his own candid revelations in the acknowledgement section, Romein admits that he was still full of ideas about improving the algorithms when the harsh reality of writing up the thesis struck. It was a task that he dreaded more than tracking down the cause of a core dump. But the latter is a deterministic problem for which a solution exists, while writing is a creative and non-deterministic activity where a single page of output may require more intellectual effort than a week's worth of debugging. Despite the wise words of Thomas Edison, that the process of inventing is one percent inspiration and 99 percent transpiration, one simply cannot succeed without that essential one percent. Maybe the Indonesian dinner Romein refers to should have come earlier in his thesis project. Nevertheless, the excellent results achieved with TDS in IDA* are of great significance, and researching its applicability to two-person game tree-searching algorithms should be next on the agenda.

In closing, the rules to obtain this nice piece of research can simply be stated as:

wise_move =
find *website*,
pick up *thesis*,
print *thesis*,
enjoy.