Appendix C

Certification of the Results given in Section 7.6

Since the bounds were calculated by a computer program requiring an unusually large memory model, and programmers and compilers cannot always be trusted, we tried to confirm the results independently, using Maple as a programming language. We did not rerun the whole computation, but we used the output of the C program described in Section 7.6 after the final iteration. The program writes the last iteration vector $\mathbf{y}^{(-i)}$ that it has computed into a file. The Maple program reads this vector and uses it as an estimate $\mathbf{y}^{\text{old}}$ for the Perron-Frobenius eigenvector with $\mathbf{y}^{\text{new}} = T_{\text{for}}\mathbf{y}^{\text{old}} \approx \lambda_W \mathbf{y}^{\text{old}}$. Instead of the standard backward iteration

$$\mathbf{y}^{\text{new}}_s = \mathbf{y}^{\text{new}}_{succ_0(s)} + \mathbf{y}^{\text{old}}_{succ_1(s)}$$

(see (7.7)), we write

$$\lambda_W \mathbf{y}^{\text{old}}_s \approx \lambda_W \mathbf{y}^{\text{old}}_{succ_0(s)} + \mathbf{y}^{\text{old}}_{succ_1(s)}.$$

Hereafter, as usual, an expression such as $\mathbf{y}^{\text{new}}_{succ_0(s)}$ or $\mathbf{y}^{\text{old}}_{succ_0(s)}$ is understood as being 0 if $succ_0(s)$ does not exist. We now find a value λ_{low} with

$$\lambda_{\text{low}} \mathbf{y}^{\text{old}}_s \leq \lambda_{\text{low}} \mathbf{y}^{\text{old}}_{succ_0(s)} + \mathbf{y}^{\text{old}}_{succ_1(s)}, \tag{C.1}$$

for all states s. In matrix notation, this is written as $\lambda_{\text{low}}\mathbf{y}^{\text{old}} \leq \lambda_{\text{low}} A\mathbf{y}^{\text{old}} + B\mathbf{y}^{\text{old}}$ or $\lambda_{\text{low}}(I - A)\mathbf{y}^{\text{old}} \leq B\mathbf{y}^{\text{old}}$. We can multiply with the nonnegative matrix $(I - A)^{-1}$ and obtain

$$\lambda_{\text{low}}\mathbf{y}^{\text{old}} \leq (I - A)^{-1} B\mathbf{y}^{\text{old}} = T_{\text{back}}\mathbf{y}^{\text{old}}.$$

By Lemma 7.9 we can now conclude that $\lambda_{\text{low}} \leq \lambda_W$.

The maximum possible value of λ_{low} is simply determined by looking at every state s and solving (C.1) for λ_{low}. We have to take the minimum of

$$\frac{\mathbf{y}^{\text{old}}_{succ_1(s)}}{\mathbf{y}^{\text{old}}_s - \mathbf{y}^{\text{old}}_{succ_0(s)}} \tag{C.2}$$

over all states s. Similarly, by reversing the inequality in (C.1) and taking the maximum of the expressions (C.2), one can find an upper bound λ_{high} on λ_W. In general, these bounds turn out to be a little weaker than the bounds that are calculated from $\mathbf{y}^{\text{old}}$ and $\mathbf{y}^{\text{new}}$ by Lemma 7.9.

In implementing this, we tried to avoid the use of excessive memory. The *C* program writes the vector $\mathbf{y}$ in such a form that the Maple program simply has to scan the file sequentially. The input file for $W = 4$ is partially shown in Figure C.1(a).

```
read "procfile.maple":
init(4):
setx({{1}}, 17554423808, 1):
setx({{4}}, 5735051264, 0):
setx({{4}}, 5735051264, 1):
setx({{1}, {4}}, 7791677952, 1):
setx({{3}}, 8280601600, 0):
setx({{4}}, 5735051264, 1):
setx({{1, 4}}, 17554423808, 1):
setx({{3, 4}}, 11470102528, 0):
...
finish():
terminate():
setx := checkx:
```

(a)

```
finish():
init(4):
read "procfile.maple":
setx := checkx:
setx({{1, 2, 3, 4}}, 28618452992, 0):
setx({{1, 2, 3, 4}}, 28618452992, 1):
setx({{1, 2, 3, 4}}, 28618452992, 1):
setx({{1, 2, 3}}, 28618452992, 0):
setx({{1, 2, 3}}, 28618452992, 1):
setx({{1, 2, 3}}, 28618452992, 1):
setx({{1, 2, 3}}, 28618452992, 1):
setx({{1, 2, 4}}, 23849553920, 0):
...
terminate():
```

(b)

Figure C.1: (a) The check file for $W = 4$. (b) The sorted version of this file.

At the start the program reads function definitions from the file procfile.maple, which is listed in Appendix D, and performs some initializations in the procedure *init*. The main work is done in the procedure calls $setx(s, y, flag)$, which indicates that the component for state s in the vector $\mathbf{y}^{\text{old}}$ equals y. A state is represented by its signature, as a set of sets. A *flag* value of 1 indicates that the program should just remember this value. A *flag* of 0 in a procedure call with state s indicates that the program should use this value and the previously-stored values of $\mathbf{y}^{\text{old}}$ to evaluate (C.2) for this state, and update λ_{low} and λ_{high} if appropriate. At the end, the procedure *finish* prints out the final values of λ_{low} and λ_{high}.

The C program writes the values for $succ_0(s)$ (if it exists) and $succ_1(s)$, with a flag of 1, immediately before writing a line for s with a flag of 0. After processing a line with a flag of 0, Maple can, therefore, forget all values that it has stored. Accordingly, as can be seen in the example of Figure C.1(a), some states occur several times.

The Maple program calculates $succ_0(s)$ and $succ_1(s)$ on its own, and it generates an error message if the required values were not stored in the preceding calls to *setx*.

We also performed a slightly more paranoid check to ensure that no state was omitted, and the program did not inadvertently use two different values for the same state. More precisely, we checked that all states that are present in the file with a flag of 1 are also present in the file with a flag of 0, with identical values y. The format of the input file is designed in such a way that one just has to sort the lines alphabetically and read the sorted file into Maple to carry out this check. Figure C.1(b) shows the sorted version of the file of Figure C.1(a)[1]. All calls to *setx* that refer to the same state are now grouped together. There must first be a call with flag 0, where the value is memorized, followed by an arbitrary number of calls with the same state and with flag 1, where the program just checks if the given values coincide. The meaning

[1]We leave the question of why the first two states in alphabetic order, $\{1, 2, 3, 4\}$ and $\{1, 2, 3\}$, have the same value, to the reader to ponder.

of the procedure *setx* is changed at the beginning by the assignment *setx* := *checkx*. Note that *finish* and *init* appear before procfile.maple is read; thus, the first two lines have no effect.

In the first phase we have already checked that all successors of all states, which are present in the file with a flag of 0, are also present in the file, with a flag of 1. As a consequence, it is ensured that at least all reachable states have been processed in the first phase (provided that at least one state was processed). This is enough to establish correctness of the result.[2]

For $W = 20$ the size of the check file was about 30 gigabytes. Each pass over the file with Maple took about 20 hours, and sorting took almost five hours. (We mention these running times only to give a rough indication. We ran our program on different computers of different speeds.) It would also be feasible to check larger values of W but we did not think it was worthwhile. The procedures *finish* and *terminate* printed the following output, after reading the unsorted check file.

```
348080  10836799
------, --------, [3.967040106, 3.967082162], 3967040105*10^-9,
87743   2731680

    3967082162*10^-9

  142547558 configurations were checked.
```

[2]We did not check if a state appears more than once with a flag of 0, possibly even with different values. It is not difficult to work out why this does not harm the reliability of the result.