Performance Notes to the RNetLogo Package

Jan C. Thiele
Department of
Ecoinformatics, Biometrics
and Forest Growth
University of Göttingen
Germany

Abstract

RNetLogo is a flexible interface for NetLogo to R, howerver, it was noticed that the performance is slow for large datasets in some circumstances. Here, I show that these problems are solved for newer versions of RNetLogo/NetLogo, and give some specific hints for using RNetLogo in situations where runtime is critical. Specifically, I present the results of an execution time measurement study of `NLGetAgentSet` using NetLogo 4.0.5, 4.1.3, and 5.0 as well as RNetLogo 0.9.2 and 0.9.3. The results show that using NetLogo 5.0 is highly recommended since its transformation times/list operations are substantially faster than in older versions of NetLogo. Some further speed improvements (for `NLGetAgentSet` or `NLGetPatches`) can be achieved by using RNetLogo >= 0.9.3.

Keywords: NetLogo, R, agent based modelling, abm, individual based modelling, ibm, performance, execution time.

1. Preliminary Note

All tests and measurements have been performed on Windows XP Professional SP3 (32-bit) with a DELL Latitude D630 notebook with an Intel(R) Core(TM)2 Duo T9500 chipset with 2.60GHz and used RAM of 3.49 GB.

The simulations have been run only once. This does not deliver reliable performance measures, because system background processes can cover the real execution times. Normally, one would run simulations multiple times and take only the minimum execution times. But to get a rough impression of the dimensions, running one simulation should be sufficient for the purpose here.

2. Motivation

Some of the users of the RNetLogo package recognized, that the data transfer using functions like the `NLGetAgentSet` was very slow on large datasets/numbers of agents. Moreover, the processing time increased non-linear with an increasing number of datasets/agents. To make it possible to work also with large datasets/number of agents I systematically analyzed the problem and identified some reasons. This paper documents the problem analysis as well as shows how to resolve the bottleneck. It also show what have changed in RNetLogo 0.9.3 in the functions NLGetAgentSet and NLGetPatches especially regarding the return data types.

3. Changes in NLGetAgentSet and NLGetPatches

3.1. Until RNetLogo 0.9.2

The results shown in the following are produced using the Fireflies model from NetLogo's Model Library. This model is initialized with different number of flies (i.e. turtles). Afterwards the NLGetAgentSet function from the RNetLogo package is used to get variables from all flies/turtles from NetLogo into R.

With RNetLogo 0.9.2 there are two data type variants for the output of the NLGetAgentSet function available: an R list (default) and an R data.frame.

The structure of the list is as follws: In case of multiple requested agent variables: for each agent there is one list element. Each of these elements contain the requested agent variables in a vector. In case of just one requested agent variable: only a single vector with the values of the different agents instead of a list is returned.

For example (using RNetLogo 0.9.2):

```
R> #RNetLogo version:
R> print(vers)
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> NLCommand("set number 10")
R> NLCommand("setup")
```

```
R> #RNetLogo version:
R> print(vers)

$Version
[1] "0.9.2"
```

```
R> # for only one agent variable it's just a vector
R> test_t5 <- NLGetAgentSet("who", "turtles", as.data.frame=F)
```

```
R> str(test_t5)

 num [1:10] 0 1 2 3 4 5 6 7 8 9

R> is.list(test_t5)

[1] FALSE
```

```
R> # for more than one agent variable it's a list
R> # with one list element for each agent and a
R> # vector in each list element containing
R> # the values of the requested agents variables
R> test_t6 <- NLGetAgentSet(c("who", "xcor", "ycor"),
+                            "turtles", as.data.frame=F)

R> str(test_t6)
```

```
List of 10
 $ : num [1:3] 0 -35.45 9.24
 $ : num [1:3] 1 -14.1 31.1
 $ : num [1:3] 2 -14.34 6.19
 $ : num [1:3] 3 18.5 -31.1
 $ : num [1:3] 4 -31.5 -26.3
 $ : num [1:3] 5 9.68 -21.5
 $ : num [1:3] 6 -24.9 -24
 $ : num [1:3] 7 -31.4 -18.8
 $ : num [1:3] 8 -34.4 -21.1
 $ : num [1:3] 9 -10.2 -27.9
```

```
R> is.list(test_t6)
```

```
[1] TRUE
```

If we request a data.frame, we have to set the argument `as.data.frame` to `TRUE` and should submit the names for the data.frame columns in the agrument `df.col.names`.

If we request only one agent variable, again, just a single vector is returned. But if we request more than one agent variable a data.frame is constructed with the agent variables in the columns and the values of each agent in rows.

For example:

```
R> # for only one agent variable its just a vector
R> test_t1 <- NLGetAgentSet("who", "turtles", as.data.frame=T,
+                            df.col.names=c("who"))

R> str(test_t1)
```

```
 num [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
R> is.data.frame(test_t1)
```

```
[1] FALSE
```

```
R> # for more than one agent variable it's a data.frame
R> # with agent variables in columns and
R> # one row for each agent
R> test_t2 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles",
+                            as.data.frame=T, df.col.names=c("who",
+                            "xcor", "ycor"))
```

```
R> str(test_t2)

'data.frame':        10 obs. of  3 variables:
 $ who : num  0 1 2 3 4 5 6 7 8 9
 $ xcor: num  -35.4 -14.1 -14.3 18.5 -31.5 ...
 $ ycor: num  9.24 31.08 6.19 -31.14 -26.29 ...

R> is.data.frame(test_t2)

[1] TRUE
```

This pattern is the same for the `NLGetPatches` function, since its functioning is equivalent to `NLGetAgentSet`.

In the manual of RNetLogo 0.9.2 it was mentioned, that the data.frame variant is much faster when requesting more than one agent variable.

This is one of the reasons why I decided to change the default return value of `NLGetAgentSet`/ `NLGetPatches` in RNetLogo 0.9.3.

3.2. Since RNetLogo 0.9.3

Since RNetLogo 0.9.3 the argument `df.col.names` isn't available anymore because the names of the requested agent variables are used as column names of the data.frame. This prevents mistankes in the order of variables and there names. If you want to replace these names just use R's `names` function.

Because the data.frame is the default return type now, it means that the function argument `as.data.frame` is `TRUE` by default. This was done because the data.frame is the standard data type in R. But if we change `as.data.frame` to `FALSE` we are not getting the data structure as known from RNetLogo 0.9.2 but a list where the list elements are the agents variables (instead of the agents) and these list elements contain vectors with the values for the agents. Therefore, it is very similar to the data structure of the data.frame. The list elements are also named with the names of the agent variables. This new list structure is substantially faster created than the old one.

If you want to produce the default list structure known from RNetLogo 0.9.2 instead, you have to set a further argument called `agents.by.row` to `TRUE` in combination with setting `as.data.frame` to `FALSE`.

Since RNetLogo 0.9.3 you will always get the expected data type independent from the number of agent variables requested. In RNetLogo 0.9.2 we got a vector when we requested only one agent variable. Now, since RNetLogo 0.9.3, we are getting either a list or data.frame depending on what was requested. This new behavior will prevent unexpected return types for single agent requests. You will now consequently getting what you asked for.

There is also the option to get a simple vector when requesting only one agent variable. Therefore, we have to set the function argument `as.vector` to `TRUE`. The result is equivalent to the output of RNetLogo 0.9.2 when requesting one agent variable independent from the requested output data type (see above).

Maybe it's more clear to see it with an example (using RNetLogo 0.9.3):

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> NLCommand("set number 10")
R> NLCommand("setup")
```

```
R> #RNetLogo version:
R> print(vers)

$Version
[1] "0.9.3"
```

```
R> # the equivalent to the RNetLogo 0.9.2 default:
R> # the list variant for only one agent variable
R> # (but now as list not as vector)
R> test_t5 <- NLGetAgentSet("who", "turtles", as.data.frame=F, agents.by.row=T)
```

```
R> str(test_t5)

List of 10
 $ : num 0
 $ : num 1
 $ : num 2
 $ : num 3
 $ : num 4
 $ : num 5
 $ : num 6
 $ : num 7
 $ : num 8
 $ : num 9

R> is.list(test_t5)

[1] TRUE
```

```
R> # for more than one agent variable
R> test_t6 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles",
+                             as.data.frame=F, agents.by.row=T)
```

```
R> str(test_t6)
```

```
List of 10
 $ : num [1:3] 0 13.1 -18
 $ : num [1:3] 1 -30.6 15.6
 $ : num [1:3] 2 16.7 29.8
 $ : num [1:3] 3 -15.5 31.6
 $ : num [1:3] 4 -16.5 -4.93
 $ : num [1:3] 5 -23.64 -7.98
 $ : num [1:3] 6 -21.6 33.6
 $ : num [1:3] 7 3.98 -30.96
 $ : num [1:3] 8 -1.09 13.02
 $ : num [1:3] 9 7.06 -30.02
```

```
R> is.list(test_t6)
```

```
[1] TRUE
```

```
R> # now the new default: the data.frame
R> # it's a data.frame independent from
R> # the number of agent variables requested
R> test_t1 <- NLGetAgentSet("who", "turtles")
```

```
R> str(test_t1)
```

```
'data.frame':        10 obs. of  1 variable:
 $ who: num  0 1 2 3 4 5 6 7 8 9
```

```
R> is.data.frame(test_t1)
```

```
[1] TRUE
```

```
R> # with three agent variables:
R> test_t2 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles")
```

```
R> str(test_t2)
```

```
'data.frame':        10 obs. of  3 variables:
 $ who : num  0 1 2 3 4 5 6 7 8 9
 $ xcor: num  13.1 -30.6 16.7 -15.5 -16.5 ...
 $ ycor: num  -18.02 15.59 29.83 31.61 -4.93 ...
```

```
R> is.data.frame(test_t2)
```

```
[1] TRUE
```

```
R> # Next, the new list style (similar to the data.frame):
R> test_t3 <- NLGetAgentSet("who", "turtles", as.data.frame=F)
```

```
R> str(test_t3)

List of 1
 $ who: num [1:10] 0 1 2 3 4 5 6 7 8 9

R> is.list(test_t3)

[1] TRUE


R> # for three agent variables:
R> test_t4 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles", as.data.frame=F)


R> str(test_t4)

List of 3
 $ who : num [1:10] 0 1 2 3 4 5 6 7 8 9
 $ xcor: num [1:10] 13.1 -30.6 16.7 -15.5 -16.5 ...
 $ ycor: num [1:10] -18.02 15.59 29.83 31.61 -4.93 ...

R> is.list(test_t4)

[1] TRUE


R> # the old data structure for one agent variable
R> # (a simple vector):
R> test_t7 <- NLGetAgentSet("who", "turtles", as.vector=T)


R> str(test_t7)

 num [1:10] 0 1 2 3 4 5 6 7 8 9

R> is.list(test_t7)

[1] FALSE

R> is.vector(test_t7)

[1] TRUE
```

Later on, we will have a look on the performance of the `NLGetAgentSet` function depending on the different output data structures.

But first, we will look on the performance depending on the NetLogo version used in conjunction with RNetLogo.

4. NetLogo dependent performance

4.1. RNetLogo 0.9.2

NetLogo 4.1.3

In this section I will give you an impression of the performance of the `NLGetAgentSet` function in dependence of the NetLogo version used with RNetLogo.

From postings on the NetLogo mailing list like this one: http://groups.yahoo.com/group/netlogo-users/message/12919, we know that the performance of list operations can be expected to be much better in NetLogo 5.0 than in NetLogo 4.1.2. As the NetLogo primitive `sort` is used in the background of the `NLGetAgentSet` function (as well as the `NLGetPatches` function), list operations are very important for the performance of this function.

As support for NetLogo 4.0.x comes first with RNetLogo 0.9.3, we can only test NetLogo 4.1.x (here I will use 4.1.3) against NetLogo 5.0 with RNetLogo 0.9.2.

Let us start with defining a function which sets up the Fireflies model and requests all flies with different number of agent variables (t1 & t5: one variable; t2 & t6: three variables) and different returning data types (t1 & t2: data.frame; t5 & t6: list). Then, we call this function with different numbers of flies (i.e. turtles) starting with 100 and multiplying it with 2 until 409.600 turtles.

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 4.1.3"
R> NLStart(nl.path, nl.version=4, gui=F)
R> model.path <- "/models/Sample Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> f_GetAgentSet <- function(x) {
+    NLCommand(paste("set number ",x,sep=""))
+    NLCommand("setup")
+
+    t1 <- system.time(df_a_1 <- NLGetAgentSet("who", "turtles",as.data.frame=T,
+                                              df.col.names=c("who"))
+                      )[["user.self"]]
+    t2 <- system.time(df_a_3 <- NLGetAgentSet(c("who", "xcor", "ycor"),
+                                              "turtles", as.data.frame=T,
+                                              df.col.names=c("who", "xcor", "ycor"))
+                      )[["user.self"]]
+
+    t5 <- system.time(li2_a_1 <- NLGetAgentSet("who", "turtles",
+                                               as.data.frame=F)
+                      )[["user.self"]]
+    t6 <- system.time(li2_a_3 <- NLGetAgentSet(c("who", "xcor", "ycor"),
+                                               "turtles", as.data.frame=F)
+                      )[["user.self"]]
+    return(data.frame(t1,t2,t5,t6))
+ }
R> it <- c(100,200,400,800,1600,3200,6400,12800,25600,51200,102400,204800,409600)
R> times_092_NL413 <- lapply(it, function(x) {f_GetAgentSet(x)})
R> #NLQuit()
```

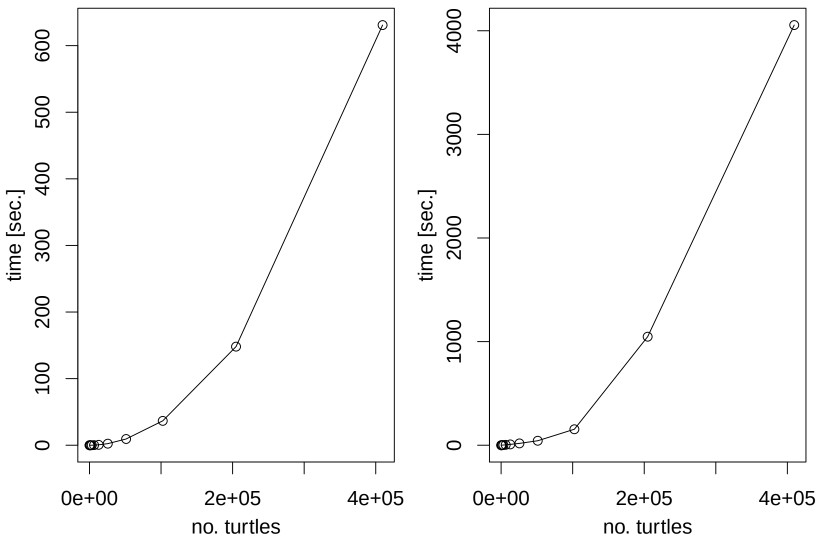

Figure 1: Execution time of `NLGetAgentSet` with list output for different number of turtles using RNetLogo 0.9.2 and NetLogo 4.1.3. Left: one agent variable, Right: three agent variables requested.

The execution time for getting the agent variables within a plain list output (one list element for each agent) with one (t5) as well as three (t6) agent variables is shown in Figure 1.

We see that the execution time increases more than linearly with an increasing number of turtles in both cases.

Now, we will have a look on the performance of the same procedure but with the data.frame as return type shown in Figure 2.

We see that the pattern is the same as with the list output type. The data.frame output type with one requested agent variable (t1) is in all cases slightly slower than the list output type. The opposite is true when three agent variables are requested (t2 vs. t6) with an exception for the last step (409.600 turtles).

NetLogo 5.0

Now, let us do the same with NetLogo 5.0.

The execution times for producing the plain list output (one list element for each agent) with one (t5) as well as three (t6) agent variables is given in Figure 3. Maybe you wonder why the execution time can be higher for smaller agentsets. If you have some experiences with NetLogo you may have seen that execution time of NetLogo strongly varies (partly due to system background process but also due to NetLogo internals). Because the execution times are very small now, we can see these variations in the plots which have been covered before by the extrem long (and increasing) operation time.

Let us have also a look on the results for the data.frame output type shown in Figure 4.

All in all, we see an extrem large reduction of the exection time by just switching from NetLogo 4.1.3 to NetLogo 5.0. For example, transforming 409.600 turtles with three agent variables into a data.frame, the execution time reduced from approximatly 4495 seconds to only 8.46

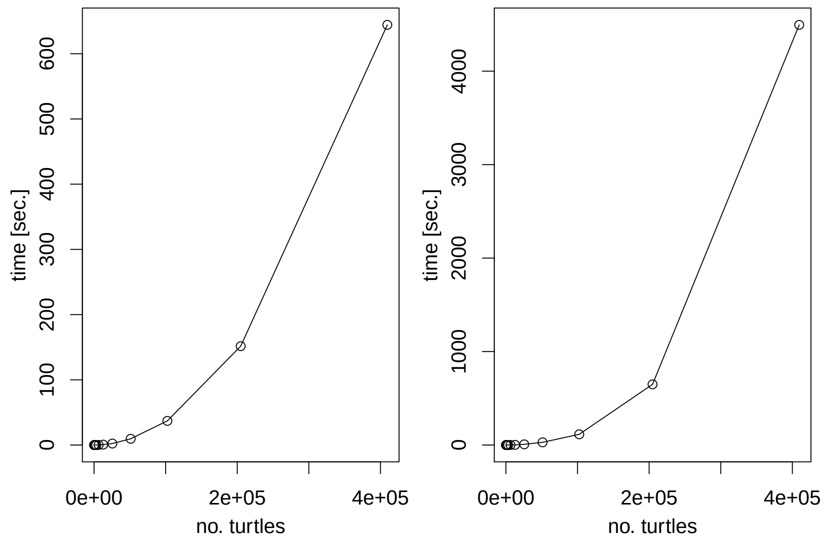

Figure 2: Execution time of `NLGetAgentSet` with data.frame output for different number of turtles using RNetLogo 0.9.2 and NetLogo 4.1.3. Left: one agent variable, Right: three agent variables requested.

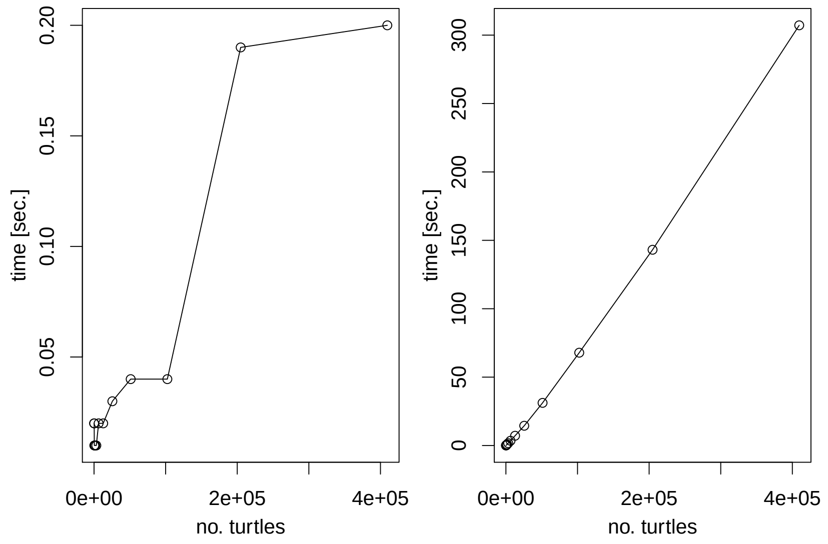

Figure 3: Execution time of `NLGetAgentSet` with list output for different number of turtles using RNetLogo 0.9.2 and NetLogo 5.0. Left: one agent variable, Right: three agent variables requested.

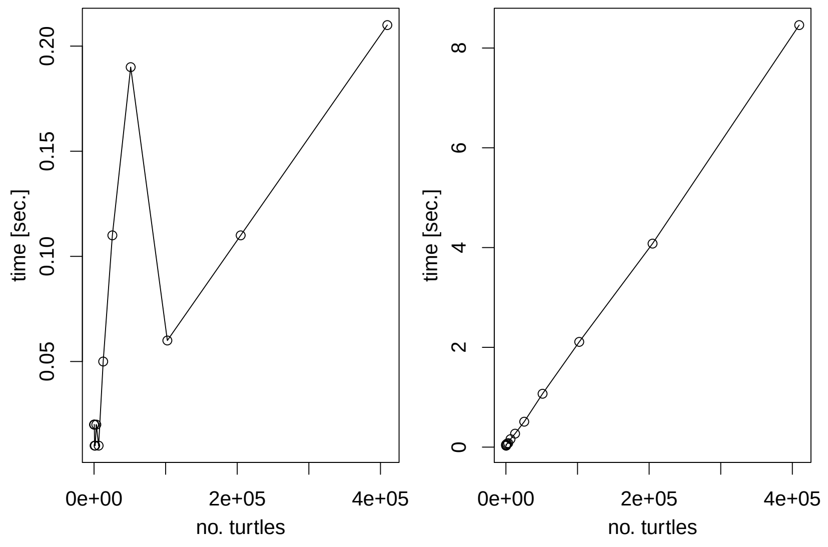

Figure 4: Execution time of `NLGetAgentSet` with data.frame output for different number of turtles using RNetLogo 0.9.2 and NetLogo 5.0. Left: one agent variable, Right: three agent variables requested.

seconds, which is a reduction of more than 530 times! This is the reason, why NetLogo 5.0 is the default version since RNetLogo 0.9.3 (argument `nl.version` in `NLStart`).

But, even this performance can be improved as you can see when switching to RNetLogo 0.9.3.

4.2. RNetLogo 0.9.3

NetLogo 5.0

Let us start with NetLogo 5.0, as we know, this is the most interesting version. At the end, we will have a short look on the comparison with NetLogo 4.1.3 and NetLogo 4.0.5.

As mentioned above, RNetLogo 0.9.3 offers three output types. The classical list output where the list elements represent the agents (which was the default until RNetLogo 0.9.2), the new list style where each list element represent an agent variable (the fastest variant for requesting multiple agent variables), and the data.frame where each agent variable is represented in a column and each agent is represented by a row (the default in NetLogo 0.9.3).

Here is the defintion of the function to iterate through different numbers of turtles:

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> f_GetAgentSet <- function(x) {
+    NLCommand(paste("set number ",x,sep=""))
+    NLCommand("setup")
+
+    t1 <- system.time(df_a_1 <- NLGetAgentSet("who", "turtles")
+                     )[["user.self"]]
+    t2 <- system.time(df_a_3 <- NLGetAgentSet(c("who", "xcor", "ycor"),
+                                             "turtles")
+                     )[["user.self"]]
+
+    t3 <- system.time(li_a_1 <- NLGetAgentSet("who", "turtles",
+                                             as.data.frame=F)
+                     )[["user.self"]]
+    t4 <- system.time(li_a_3 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles",
+                                             as.data.frame=F)
+                     )[["user.self"]]
+
+    t5 <- system.time(li2_a_1 <- NLGetAgentSet("who", "turtles", as.data.frame=F,
+                                              agents.by.row=T)
+                     )[["user.self"]]
+    t6 <- system.time(li2_a_3 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles",
+                                              as.data.frame=F,
+                                              agents.by.row=T)
+                     )[["user.self"]]
+
+    t7 <- system.time(ve_a_1 <- NLGetAgentSet(c("who", "xcor", "ycor"), "turtles",
+                                              as.vector=T)
+                     )[["user.self"]]
+    return(data.frame(t1,t2,t3,t4,t5,t6,t7))
+  }
R> it <- c(100,200,400,800,1600,3200,6400,12800,25600,51200,102400,204800,409600)
R> times_093_NL5 <- lapply(it, function(x) {f_GetAgentSet(x)})
R>
R> #NLQuit()
```

Let us start with a comparison of the performance of the three different output types for requesting one agent variable as shown in Figure 5.

We see, that the execution times for the old list style (which was the default in RNetLogo <= 0.9.2) are larger than in RNetLogo 0.9.2. The explanation for this is very easy: the old RNetLogo version returned just a vector when the user requested only one agent variable while the new version constructs the requested list which is more logical but takes more time. The same is true for the data.frame construction. Furthermore, we see that the new list style is not faster than the data.frame construction when requesting only one agent variable.

But this conclusion changes when we look on the execution times for three agent variables (Figure 6).

The time for requesting three agent variables in the old list style takes approximatly the same time with RNetLogo 0.9.3 as with the old version. But the performance of the data.frame is much better than with the old version when requesting more than one agent variable. For

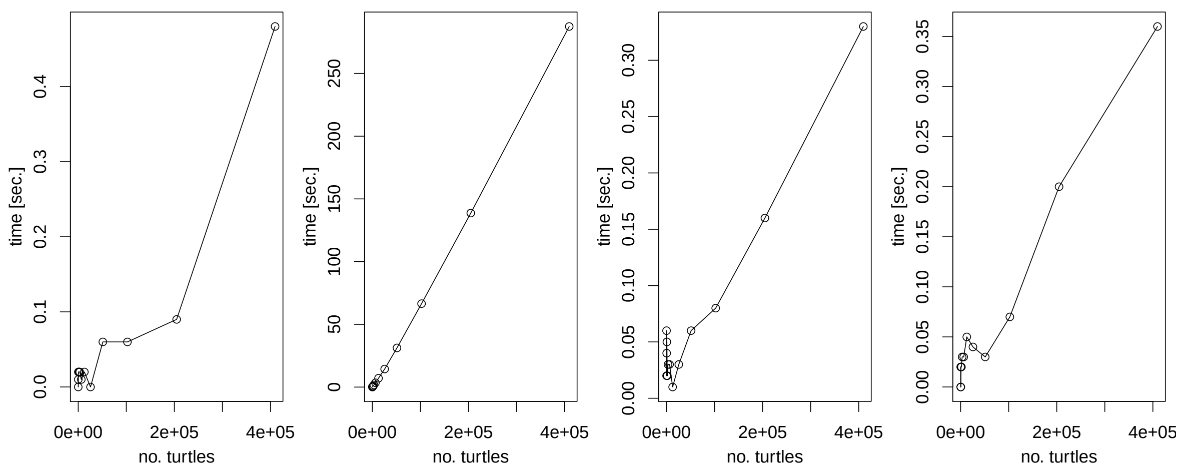

Figure 5: Execution time of `NLGetAgentSet` with one requested agent variable for different number of turtles using RNetLogo 0.9.3 and NetLogo 5.0. Left: simple vector, Second from left: old list style, Second from right: data.frame, Right: new list style.

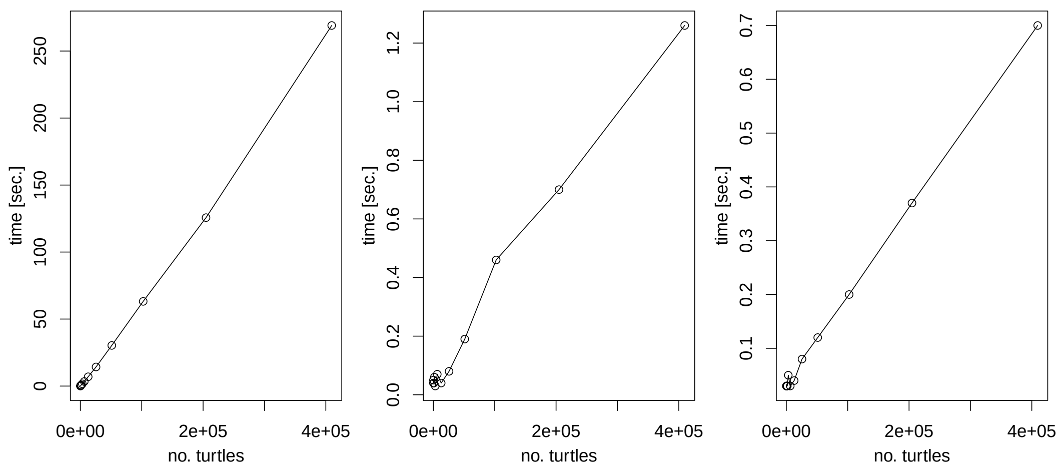

Figure 6: Execution time of `NLGetAgentSet` with three requested agent variables for different number of turtles using RNetLogo 0.9.3 and NetLogo 5.0. Left: old list style, Middle: data.frame Right: new list style.

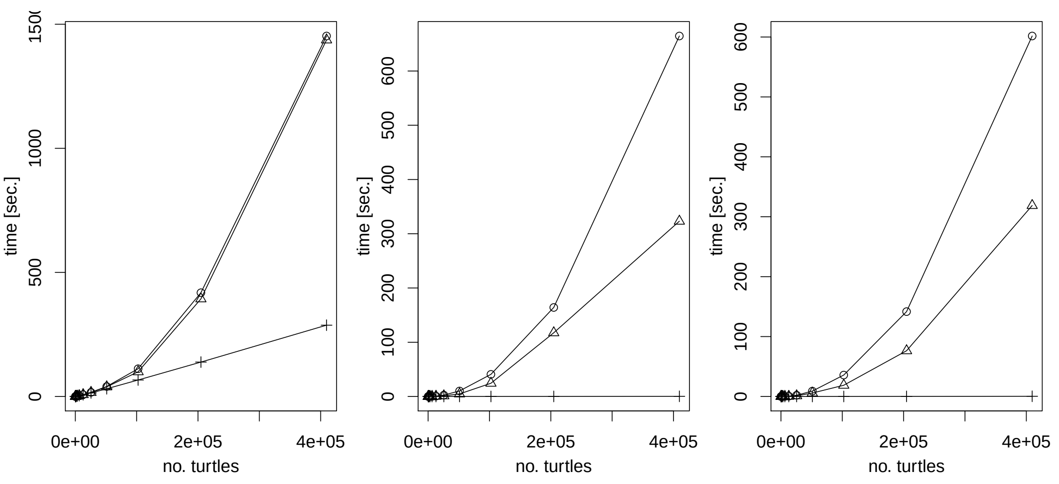

Figure 7: Execution time of `NLGetAgentSet` with one requested agent variable for different number of turtles using RNetLogo 0.9.3 and NetLogo 4.0.5 (dots), 4.1.3 (triangle), and 5.0 (plus). Left: old list style, Middle: data.frame Right: new list style.

409.600 turtles with three variables it took approximatly 8.46 seconds with RNetLogo 0.9.2 and now, with RNetLogo 0.9.3, it takes approximatly 1.26 seconds, which is a reduction of more than six times. Furthermore, we see that the new list style is even slightly faster than the data.frame with only 0.7 seconds for 409.600 turtles with three variables (i.e. the transformation of 1.228.800 values!).

NetLogo 5.0 vs. 4.1.3 vs. 4.0.5

Just for interest, we will compare also the performance of NetLogo 4.1.3 with the one of NetLogo 4.0.5, which is possible since RNetLogo 0.9.3, as well as with NetLogo 5.0.

The results for only one agent variable are shown in Figure 7 and the results for requesting three agent variables are given in Figure 8.

In (nearly) all cases, NetLogo 4.0.5 performs better than NetLogo 4.1.3 but it is also far behind NetLogo 5.0.

5. Conclusion

We have seen that, whenever possible, NetLogo 5.0 should be prefered over NetLogo 4.x. The performance of `NLGetAgentSet` is substantially better with NetLogo 5.0. These results are representive for other operations/functions. Furthermore, RNetLogo 0.9.3 performs better than older versions when requesting more than one agent variable with the `NLGetAgentSet` or `NLGetPatches` functions. The old list style should be avoided. The data.frame is now the default return type and performs well. But if it is sufficient to proceed with a list, the new list style is even faster than the data.frame and should be prefered.

With NetLogo 5.0 and RNetLogo 0.9.3, in most cases the data transformation from NetLogo to R should be fast enough even for requesting large numbers of turtles or patches. To get three agent variables for 409.600 turtles (i.e. 1.228.800 values) from NetLogo into R took in the

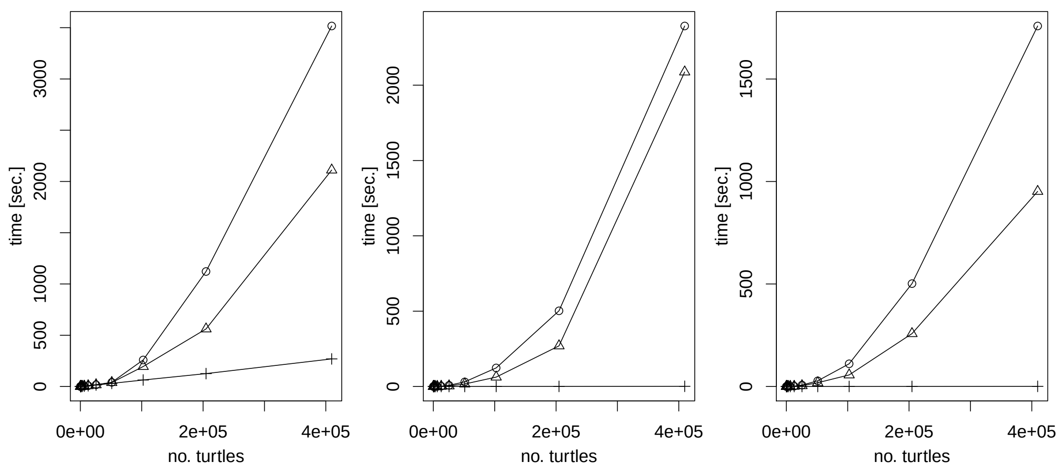

Figure 8: Execution time of `NLGetAgentSet` with three requested agent variables for different number of turtles using RNetLogo 0.9.3 and NetLogo 4.0.5 (dots), 4.1.3 (triangle), and 5.0 (plus). Left: old list style, Middle: data.frame Right: new list style.

example only 1.26 seconds for a data.frame and 0.7 seconds for the new list style. The setup procedure in the Fireflies model in pure NetLogo (without using RNetLogo and switching off the visual update) with 409.600 turtles took for example approximatly 61 seconds and the execution of one simulation step (go procedure) with this number of turtles took approximatly 80 seconds. In conclusion, I think you will find RNetLogo isn't a bottleneck.

Acknowledgement

Thanks go to Florian Hartig (UFZ - Helmholtz Centre for Environmental Research, Leipzig) and Thomas Petzold (University of Dresden) who found it took very long to transform large agentsets and motivated me to look closer on this topic. Furthermore, Florian made some valueable notes on this manuscript.

Affiliation:
Jan C. Thiele
Department of Ecoinformatics, Biometrics and Forest Growth
University of Göttingen
Büsgenweg 4
37077 Göttingen, Germany
E-mail: `jthiele@gwdg.de`
URL: `http://www.uni-goettingen.de/en/72779.html`