Package 'FLSSS'

April 20, 2016

Type Package

Title Multithreaded Fixed Size Subset Sum with Bounded Error in Multidimensional Real Domain

Version 5.0.1

Date 2016-04-18

Author Charlie Wusuo Liu

Maintainer Charlie Wusuo Liu <liuwusuo@gmail.com>

Description A novel algorithm for solving the fixed size Subset Sum Problem with bounded error in multidimensional real domain.

License GPL-2

Imports Rcpp,RcppParallel

LinkingTo Rcpp,RcppParallel

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-04-20 08:53:22

R topics documented:

`FLSSS` *Single threaded fixed size Subset Sum with bounded error in one-dimensional real domain*

Description

Given subset size "len", sorted numeric set "v", subset-sum "target", error "ME", find one or more subsets of size len whose elements sum into [target - ME, target + ME]

Usage

```
FLSSS(len, v, target, ME, sizeNeeded = 1L,
tlimit="none", catch=NULL, throw=F, LB=1L:len,
UB=(length(v)-len+1L):length(v))
```

Arguments

len	Subset size
v	Sorted numeric vector as the set
target	Subset-sum
ME	Error
sizeNeeded	How many solutions at least are wanted; sizeNeeded="all" returns all the solutions
tlimit	Time Limit in seconds. E.g. tlimit=10 let FLSSS stop to return all the solutions found in 10 seconds. tlimit="none" means no time limit
catch	Catch the stack object implemented as a list
throw	Throw the stack object as a list
LB	Lower bound initializer
UB	Upper bound initializer

Details

An introduction to the algorithm

C++ and R codes for version 1.0

Explanatory R codes

Additional C++ codes

Value

throw=F, returns a list of integer vectors. Each vector contains a solution's indexes.

throw=T, returns a list L, L$roots is the list of solutions, L$node is the list of the stack.

Note

1. Parameter "catch" and "throw": for example, if sizeNeeded=10 and throw=1, FLSSS will return a list L where L$roots are the solutions and L$node is a "stack" list. If 10 more solutions are needed then FLSSS with catch=L$node will start looking for the 11th solution directly.

2. Number of output solutions may be greater than sizeNeeded. A resulting empty list indicates no solutions. Setting sizeNeeded="all" and tlimit="none" could take considerable amount of time when v is long and sparse. Assigning at least one of them numeric value is strongly recommended.

3. When v is real and an error of 0 is expected, it is recommended to set ME=0.01, 0.001 or whatever less than the decimal point v's elements have reached.

Examples

```
v<-c(-1119924501,-793412295,-496234747,-213654767,16818148,
26267601,26557292,27340260,28343800,32036573,32847411,
34570996,34574989,43633028,44003100,47724096,51905122,
52691025,53600924,56874435,58207678,60225777,60639161,
60888288,60890325,61742932,63780621,63786876,65167464,
66224357,67198760,69366452,71163068,72338751,72960793,
73197629,76148392,77779087,78308432,81196763,82741805,
85315243,86446883,87820032,89819002,90604146,93761292,
97920291,98315039,310120088)
len<-10
target<-139254955
FLSSS(len,v,target,0,"all")
FLSSS(len,v,target,10,"all",tlimit=2)
FLSSS(len,v,target,20,5)

result1=FLSSS(len,v,target,20,"all",tlimit=0.03,throw=1)
result2=FLSSS(len,v,target,20,"all",tlimit=0.1,catch=result1$node)

# Example of using FLSSS to solve Subset Sum Problem
for(len in 1:length(v))
{
  lis=FLSSS(len,v,target,0)
  if(length(lis)!=0)break
}

# Example of using FLSSS to solve a knapsack problem:

bagCapacity=361
# Capacity of the knapsack

objectMass<-sort(sample(1:100,10))
# Mass of each of the 10 kinds of objects that can
# be put in the knapsack

objectAmount<-sample(1:30,10)
# Number of each kind

v<-unlist(mapply(rep,objectMass,objectAmount))
# Set of choices

for(len in 1:length(v))
{
  lis=FLSSS(len,v,bagCapacity,0)
  # Use up the capacity of the knapsack with the smallest number of objects

  if(length(lis)!=0)break
}
```

mFLSSSpar *Multithreaded fixed size Subset Sum with bounded error in multidimensional real domain*

Description

FLSSS expanded to multidimensional space with parallel computing option

Usage

```
mFLSSSpar(len, mV, mTarget, mME, maxCore = 8L,
totalSolutionNeeded = 1L, tlimit = 60,
singleSolutionNeeded = 1L, randomizeTargetOrder=T,
LB = 1L:len, UB = (nrow(mV) - len + 1L):nrow(mV))
```

Arguments

len	An integer as the subset size
mV	A data frame as the multidimensional numeric vector/set. Each row is an element of the vector/set.
mTarget	A numeric vector as the subset-sum for each dimension. Its length should equal the number of columns/dimensions in mV.
mME	A numeric vector as the error for each dimension. Should have the same length as mTarget.
maxCore	Number of threads to invoke. Better not be greater than the available CPUs on machine. Number of cores can be found by your operating system's Task Manager.
totalSolutionNeeded	Number of solutions needed.
tlimit	Time limit in seconds. Default to 60s
singleSolutionNeeded	Default to 1. Discussed in the Details section.
randomizeTargetOrder	Default to TRUE. Discussed in the Details section.
LB	Lower bound initializer.
UB	Upper bound initializer.

Details

Since FLSSS is a comparison-based algorithm, it's intuitive to expand all its arithemetic operations to multidimensional reals to solve a multidimensional fixed size Subset Sum problem, denoted by MFSSS. However, this requires all the dimensions in mV(the multidimensional vector/set) are comonotonic, i.e., perfectly positively rank-correlated. Denote this overloaded algorithm "mFLSSS".

An algorithm is invented to comonotonize the dimensions. To solve MFSSS appropriately, it will trigger no more than k*(N-k)+1 calls for mFLSSS (N is the length of the multidimensional vector

and k is the subset size), each time with a different subset-sum target. The parameter "singleSolutionNeeded", equivalent to "sizeNeeded" in FLSSS, feeds mFLSSS each time when it's being called.

The no more than k*(N-k)+1 subset-sum targets are independent of mFLSSS, so the sequence of mFLSSS calls are parallelizable. The final algorithm is written in a multithreaded fashion, denoted by "mFLSSSpar".

In mFLSSSpar, each thread will communicate with its peers and track the total number of solutions that have been mined. Any thread that finds parameter "totalSolutionNeeded" reached will have all the threads quit immediately.

To gain performance, the order of the no more than k*(N-k)+1 subset-sum targets are randomized by default parameter "randomizeTargetOrder", so mFLSSSpar could give different solutions and performances each run with the same input. The targets are ordered ascendingly if randomizeTargetOrder is set FALSE.

Value

A list of integer vectors. Each vector is a solution's indexes.

Note

Unlike single-dimensional fixed size Subset Sum which "rarely contains no solution" in the entire combinatorial, the multidimensional fixed size Subset Sum "rarely contains a solution" when the number of dimensions is not trivial.

This fact could make a seemingly trivial task in FLSSS pretty heavy in mFLSSSpar, considering only the sizes of the subset and superset. It is recommended to run mFLSSSpar with smaller subset size or larger error a few times to learn its performance if you are dealing with non-trivial task.

Examples

```
len=6L
# subset size

d=5L
# number of dimensions

lenv=100L
# size of the vector/set

v=as.data.frame(matrix(rnorm(d*lenv)*1000,ncol=d))
# generate a multidimensional vector/set

solution=sample(1L:lenv, len)
# make an underlying solution

target=as.numeric(colSums(v[solution,]))
# the subset-sum target of the underlying solution

ME=abs(target)*0.05
# bound the error as 5% of each dimension's magnitude
```

```
tmp=mFLSSSpar(len,v,target,ME)
# try finding at least one solution within a minute

if(length(tmp)!=0)abs(as.numeric(colSums(v[tmp[[1]],]))/target-1)
# exam the solution
```

Index