Report index

StatProfilerHTML.jl report

Generated on Tue 31 May 2022 03:17:20 PM -03

File source code

Line	Exclusive	Inclusive	Code
1			function classicBPseq(NMRdata::NMRType,
2			ε :: Float64,
3			virtual_c :: Float64,
4			allmol :: Bool,maxl=25,bug=true)
5			if allmol == true
6			error("This solver is not prepared to find all solutions yet")
7			end
8			n = NMRdata.dim
9			mol = MoleculeType(Vector{AtomType}(undef,n),0.0)
10			for i=1:n
11			mol.atoms[i] = AtomType(0.0,0.0,0.0)
12			end
13			nsol = 0
14			storage_mol = Dict{Int64,MoleculeType}()
15			n_prune = 0
16			n_branch = 0
17			#count_nop = [+-,*,/,√]
18			nop_node = [0,0,0,0]
19			nop_ddf = [0,0,0,0]
20			nop_vpath = [0,0,0,0]
21			# first atom
22			mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
23			mol.atoms[1].x = 0.0
24			mol.atoms[1].y = 0.0
25			mol.atoms[1].z = 0.0
26			#second atom
27			mol.atoms[2].element = NMRdata.info[2,:].nzval[1].atom1
28			mol.atoms[2].x = -NMRdata.info[1,2].dist
29			mol.atoms[2].y = 0.0
30			mol.atoms[2].z = 0.0
31			# tird atom
32			D12 = NMRdata.info[1,2].dist
33			D13 = NMRdata.info[1,3].dist
34			D23 = NMRdata.info[2,3].dist
35			D14 = 0.0
36			D24 = 0.0
37			D34 = 0.0
38			c0,s0 = bondangle(D12,D13,D23)
39			nop_node += [3,6,1,1]
40			cw,sw = (0.0,0.0)
41			mol.atoms[3].element = NMRdata.info[3,:].nzval[1].atom1
42			mol.atoms[3].x = -D12+D23*c0
43			mol.atoms[3].y = D23*s0
44			mol.atoms[3].z = 0.0
45			nop_node += [1,2,0,0]
46			C = zeros(4,4)
47			C[1,4] = mol.atoms[3].x
48			C[2,4] = mol.atoms[3].y
49			C[3,4] = mol.atoms[3].z
50			C[1,1] = c0
51			C[1,2] = s0
52			C[2,1] = s0
53			C[2,2] = -c0
54			C[3,3] = -1.0
55			C[4,4] = 1.0
56			
57			l = 4 # branching starts at atom 4
58			pos = 4 # position in virtual path
59			explore_right_side = zeros(Bool,n)
60			C_list = Array{Array{Float64,2}}(undef,n) # to acess level l it is need to put l-3
61			# println(C)
62			C_list[3] = copy(C)
63			# C_before = zeros(4,4)
64			B = Array{Array{Float64,2}}(undef,n) # to acess level l it is need to put l-3
65			B[4] = zeros(4,4)
66			first_ocor = -1*ones(Int,n)
67			virtual_count = 0
68			while l<=n && l>3
69			#if 4<l <= maxl
70			# println("B matrix in level $(l-1) ")
71			# display(B[l-1])
72			# println("C matrix")
73			# display(C_list[l-1])
74			#end
75			
76			#display(l)
77			
78			if l==maxl+1
79			error("bla")
80			end
81			# if first_ocor[l-1] == -1 && first_ocor[NMRdata.virtual_path[l-1]] == -1
82			# first_ocor[l-1] = NMRdata.virtual_path[l-1]
83			# end
84			# TODO: otimizar!
85			pos = findfirst(x->x==l-1,NMRdata.virtual_path) +1
86			#C_before = zeros(4,4)
87			C_before = copy(C_list[l-1])
88			keep = true
89			while keep
90			try
91			D14 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos]].dist
92			catch
93			error("${[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos]]}")
94			D14 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
95			end
96			try
97			D24 = NMRdata.info[NMRdata.virtual_path[pos-2],NMRdata.virtual_path[pos]].dist
98			catch
99			D24 = sqrt((mol.atoms[NMRdata.virtual_path[pos-2]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-2]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-2]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
100			end
101			try
102			D34 = NMRdata.info[NMRdata.virtual_path[pos-1],NMRdata.virtual_path[pos]].dist
103			catch
104			D34 = sqrt((mol.atoms[NMRdata.virtual_path[pos-1]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-1]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-1]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
105			end
106			try
107			D12 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-2]].dist
108			catch
109			D12 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos-2]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos-2]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos-2]].z)^2)
110			end
111			try
112			D13 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-1]].dist
113			catch
114			D13 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos-1]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos-1]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos-1]].z)^2)
115			end
116			try
117			D23 = NMRdata.info[NMRdata.virtual_path[pos-2],NMRdata.virtual_path[pos-1]].dist
118			catch
119			D23 = sqrt((mol.atoms[NMRdata.virtual_path[pos-2]].x - mol.atoms[NMRdata.virtual_path[pos-1]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-2]].y - mol.atoms[NMRdata.virtual_path[pos-1]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-2]].z - mol.atoms[NMRdata.virtual_path[pos-1]].z)^2)
120			end
121			c0,s0 = bondangle(D23,D24,D34)
122			cw,sw = badtorsionangle(D12,D13,D14,D23,D24,D34)
123			# println("l value = $(l) and NMRdatavalue = $(NMRdata.virtual_path[pos]) in position $(pos)")
124			# @show c0, s0,cw,sw, D34
125			
126			
127			B[l] = torsionmatrix(c0,s0,cw,sw,D34)
128			# if explore_right_side[l] == true
129			# B[l] = torsionmatrix(B[l])
130			# end
131			if l==NMRdata.virtual_path[pos]
132			nop_node += [0,7,0,0] #torsion matrix
133			nop_node += [3,6,1,1] # bond angle
134			nop_node += [10,20,4,2] # bad torsion angle

```
                    C_list[1] = prodmatrix(C_before,B[1])
                    nop_node += [24,33,0,0]
                    keep = false

                else
                    nop_node += [0,7,0,0] #torsion matrix
                    nop_node += [3,6,1,1] # bond angle
                    nop_node += [10,20,4,2] # bad torsion angle

                    nop_vpath += [0,7,0,0] # torsion matrix
                    nop_vpath += [3,6,1,1] # bond angle
                    nop_vpath += [10,20,4,2] # bad torsion angle

                    cpx = mol.atoms[NMRdata.virtual_path[pos]].x
                    cpy = mol.atoms[NMRdata.virtual_path[pos]].y
                    cpz = mol.atoms[NMRdata.virtual_path[pos]].z
                    if bug
                        println("($cpx , $cpy ,$cpz )")
                    end
                    Virtual_Torsion = prodmatrix(C_before,B[1])
                    # println("Virtual Torsion = $(C_before) * $(B[1])$(Virtual_Torsion)")
                    nop_vpath += [24,33,0,0]
                    nop_node += [23,33,0,0]

                    if sqrt((Virtual_Torsion[1,4]- cpx)^2+(Virtual_Torsion[2,4]- cpy)^2+(Virtual_Torsion[3,4]- cpz)^2)> virtual_ε
                        B[1] = torsionmatrix(B[1])
                        C_before = prodmatrix(C_before,B[1])
                        nop_vpath += [24,33,0,0]
                        nop_node += [24,33,0,0]
                        #    println("passou 1")
                    else
                        #println("passou 2")
                        C_before = copy(Virtual_Torsion)
                        #C_before = Virtual_Torsion
                    end
                    if bug
                        println("Virtual Torsion")
                        display(Virtual_Torsion)
                    end
                    #println("Torsion matrix $(B[1])")
                    #println("C_before matrix $(C_before)")
                    #@debug "virtual atom position  " C_before[1,4],C_before[2,4],C_before[3,4]
                    pos = pos+1
                end
            end


            if explore_right_side[l] == false
                if bug && (3<l<= maxl)
                    println("B matrix in level $(l) in left side")
                    display(B[l])
                    println("C matrix ")
                    display(C_list[l])
                end
                mol.atoms[l].element = NMRdata.info[1,:].nzval[1].atom1
                mol.atoms[l].x = C_list[l][1,4]
                mol.atoms[l].y = C_list[l][2,4]
                mol.atoms[l].z = C_list[l][3,4]
                count = [0,0,0,0]
                λ , count  = pruningtest(mol,l,NMRdata,ε,count)
                nop_ddf += count
                #println("C = C_before*B at level $(l) left side  $(C_list[l])) = $(C_list[l-1]) * $(B[l])")
                if λ == 1
                    if l<n
                        #        println("Partial solution by left side at level $(l) " , mol)
                        n_branch +=1
                        #classicBP_closure(l+1,pos+1,mol,C)
                        #explore_right_side[l+1]=false
                    else
                        nsol=nsol+1
                        storage_mol[nsol] = copy(mol)
                        @debug "Rank n was reached, a solution was found "
                    end
                else
                    n_prune += 1
                    explore_right_side[l] = true
                end
            end
            if explore_right_side[l] == true
                #display(l)
                B[l] = torsionmatrix(B[l])
                #nop_node += [0,0,0,0]
                C_list[l] = prodmatrix(C_before,B[l])# tenho que otimizar este calculo
                if bug && (3<l<=maxl)
                    println("B matrix in level $(l) in right side")
                    display(B[l])
                    println("C matrix")
                    display(C_list[l])
                end
                nop_node += [24,33,0,0]
                mol.atoms[l].x = C_list[l][1,4]
                mol.atoms[l].y = C_list[l][2,4]
                mol.atoms[l].z = C_list[l][3,4]
                count = [0,0,0,0]
                ρ ,count = pruningtest(mol,l,NMRdata,ε,count) #preciso modificar
                nop_ddf += count
                #println("C = C_before*B at level $(l) right side  $(C_list[l])) = $(C_list[l-1]) * $(B[l])")
                if ρ == 1
                    if l<n
                        #        println("Partial solution by right side at level $(l) ", mol)
                        n_branch += 1
                        #explore_right_side[l+1]=false
                        #classicBP_closure(l+1,pos+1,mol,C)
                    else
                        nsol = nsol+1
                        storage_mol[nsol] = copy(mol)
                        @debug "Rank n was reached, a solution was found "
                    end
                else
                    explore_right_side[l] = false
                    k = l-1
                    while explore_right_side[k] == true
                        explore_right_side[k] = false
                        k -= 1
                    end
                    explore_right_side[k] = true
                    l = k-1
                    n_prune += 1
                end


            end
            l += 1
        end
        if l == 3
            error("Solution not found, problem possible infeasible")
        end
        display(first_ocor)
        return nsol, storage_mol,Counter(nop_node,nop_vpath,nop_ddf,n_branch,n_prune)
    end

    ###############
    function classicBPseq2(NMRdata::NMRType,
            ε :: Float64,
            virtual_ε :: Float64,
            allmol :: Bool)
        if allmol == true
            error("This solver is not prepared to find all solutions yet")
        end

        n = NMRdata.dim
```

```
mol = MoleculeType(Vector{AtomType}(undef,n),0.0)
for i=1:n
    mol.atoms[i] = AtomType(0.0,0.0,0.0)
end
C = zeros(4,4)
nsol = 0
storage_mol = Dict{Int64,MoleculeType}()
n_prune = 0
n_branch = 0
#count_nop = [+-,*,/,√]
nop_node = [0,0,0,0]
nop_ddf = [0,0,0,0]
nop_vpath = [0,0,0,0]
#1 = 1 first atom  # first atom
mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
mol.atoms[1].x = 0.0
mol.atoms[1].y = 0.0
mol.atoms[1].z = 0.0
#second atom
mol.atoms[2].element = NMRdata.info[2,:].nzval[1].atom1
mol.atoms[2].x = -NMRdata.info[1,2].dist
mol.atoms[2].y = 0.0
mol.atoms[2].z = 0.0
# tird atom
D12 = NMRdata.info[1,2].dist
D13 = NMRdata.info[1,3].dist
D23 = NMRdata.info[2,3].dist
D14 = 0.0
D24 = 0.0
D34 = 0.0
c0,s0 = bondangle(D12,D13,D23)
nop_node += [3,6,1,1]
cw,sw = (0.0,0.0)
mol.atoms[3].element = NMRdata.info[3,:].nzval[1].atom1
mol.atoms[3].x = -D12+D23*c0
mol.atoms[3].y = D23*s0
mol.atoms[3].z = 0.0
nop_node += [1,2,0,0]
C = zeros(4,4)
C[1,4] = mol.atoms[3].x
C[2,4] = mol.atoms[3].y
C[3,4] = mol.atoms[3].z
C[1,1] = c0
C[1,2] = s0
C[2,1] = s0
C[2,2] = -c0
C[3,3] = -1.0
C[4,4] = 1.0
l = 4 # branching starts at atom 4
pos = 4 # position in virtual path
explore_right_side = zeros(Bool,n)
C_list = Array{Array{Float64,2}}(undef,n) # to acess level l it is need to put l-3
println(C)
C_list[3] = C
C_before = zeros(4,4)
B = Array{Array{Float64,2}}(undef,n) # to acess level l it is need to put l-3
B[4] = zeros(4,4)
while l<=n
    pos = findall(NMRdata.virtual_path.==l-1)[1] +1
    display(l)
    copyto!(C_before,C_list[l-1])
    keep = true
    while keep
        try
            D14 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos]].dist
        catch
            D14 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
        end
        try
            D24 = NMRdata.info[NMRdata.virtual_path[pos-2],NMRdata.virtual_path[pos]].dist
        catch
            D24 = sqrt((mol.atoms[NMRdata.virtual_path[pos-2]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-2]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-2]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
        end
        try
            D34 = NMRdata.info[NMRdata.virtual_path[pos-1],NMRdata.virtual_path[pos]].dist
        catch
            D34 = sqrt((mol.atoms[NMRdata.virtual_path[pos-1]].x - mol.atoms[NMRdata.virtual_path[pos]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-1]].y - mol.atoms[NMRdata.virtual_path[pos]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-1]].z - mol.atoms[NMRdata.virtual_path[pos]].z)^2)
        end
        try
            D12 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-2]].dist
        catch
            D12 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos-2]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos-2]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos-2]].z)^2)
        end
        try
            D13 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-1]].dist
        catch
            D13 = sqrt((mol.atoms[NMRdata.virtual_path[pos-3]].x - mol.atoms[NMRdata.virtual_path[pos-1]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-3]].y - mol.atoms[NMRdata.virtual_path[pos-1]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-3]].z - mol.atoms[NMRdata.virtual_path[pos-1]].z)^2)
        end
        try
            D23 = NMRdata.info[NMRdata.virtual_path[pos-2],NMRdata.virtual_path[pos-1]].dist
        catch
            D23 = sqrt((mol.atoms[NMRdata.virtual_path[pos-2]].x - mol.atoms[NMRdata.virtual_path[pos-1]].x)^2 + (mol.atoms[NMRdata.virtual_path[pos-2]].y - mol.atoms[NMRdata.virtual_path[pos-1]].y)^2+ (mol.atoms[NMRdata.virtual_path[pos-2]].z - mol.atoms[NMRdata.virtual_path[pos-1]].z)^2)
        end
        c0,s0 = bondangle(D23,D24,D34)
        cw,sw = badtorsionangle(D12,D13,D14,D23,D24,D34)
        println("l value = $(l) and NMRdatavalue = $(NMRdata.virtual_path[pos]) in position $(pos)")
        B[l] = torsionmatrix(c0,s0,cw,sw,D34)
        # if explore_right_side[l] == true
        #    B[l] = torsionmatrix(B[l])
        # end
        if l==NMRdata.virtual_path[pos]
            nop_node += [0,7,0,0] #torsion matrix
            nop_node += [3,6,1,1] # bond angle
            nop_node += [10,20,4,2] # bad torsion angle
            C_list[l] = prodmatrix(C_list[l-1],B[l])
            nop_node += [24,33,0,0]
            keep = false
        else
            nop_node += [0,7,0,0] #torsion matrix
            nop_node += [3,6,1,1] # bond angle
            nop_node += [10,20,4,2] # bad torsion angle

            nop_vpath += [0,7,0,0] # torsion matrix
            nop_vpath += [3,6,1,1] # bond angle
            nop_vpath += [10,20,4,2] # bad torsion angle

            cpx = mol.atoms[NMRdata.virtual_path[pos]].x
            cpy = mol.atoms[NMRdata.virtual_path[pos]].y
            cpz = mol.atoms[NMRdata.virtual_path[pos]].z
            Virtual_Torsion = prodmatrix(C_list[l-1],B[l])
            println("Matriz Virtual Torsion")
            display(Virtual_Torsion)
            # println("Virtual Torsion = $(C_before) * $(B[l])$(Virtual_Torsion)")
            nop_vpath += [24,33,0,0]
            nop_node += [23,33,0,0]

            if sqrt((Virtual_Torsion[1,4]- cpx)^2+(Virtual_Torsion[2,4]- cpy)^2+(Virtual_Torsion[3,4]- cpz)^2)> virtual_ε
                B[l] = torsionmatrix(B[l])
                C_list[l-1] = prodmatrix(C_list[l-1],B[l])
                nop_vpath += [24,33,0,0]
                nop_node += [24,33,0,0]
                #println("passou 1")
            else
                #println("passou 2")
                copyto!(C_list[l-1],Virtual_Torsion)
                #C_before = Virtual_Torsion
            end
            #println("Torsion matrix $(B[l])")
            #println("C_before matrix $(C_before)")
            #@debug "virtual atom position  " C_before[1,4],C_before[2,4],C_before[3,4]
            pos = pos+1
```

417			end
418			end
419			
420			
421			if explore_right_side[1] == true
422			B[1] = torsionmatrix(B[1])
423			#nop_node += [0,0,0,0]
424			C_list[1] = prodmatrix(C_list[1-1],B[1])# tenho que otimizar este calculo
425			nop_node += [24,33,0,0]
426			end
427			mol.atoms[1].x = C_list[1][1,4]
428			mol.atoms[1].y = C_list[1][2,4]
429			mol.atoms[1].z = C_list[1][3,4]
430			count = [0,0,0,0]
431			p ,count = pruningtest(mol,1,NMRdata,ε,count) #preciso modificar
432			nop_ddf += count
433			#println("C = C_before*B at level $(1) right side $(C_list[1])) = $(C_list[1-1]) * $(B[1])")
434			if p == 1
435			if l<n
436			# println("Partial solution by right side at level $(1) ", mol)
437			n_branch += 1
438			l += 1
439			else
440			nsol = nsol+1
441			storage_mol[nsol] = copy(mol)
442			@debug "Rank n was reached, a solution was found "
443			end
444			else
445			if explore_right_side[1] == true
446			explore_right_side[1] = false
447			k = 1-1
448			while explore_right_side[k] == true
449			explore_right_side[k] = false
450			k -= 1
451			end
452			explore_right_side[k] = true
453			l = k
454			n_prune += 1
455			else
456			explore_right_side[1] = true
457			end
458			end
459			end
460			return nsol, storage_mol,Counter(nop_node,nop_vpath,nop_ddf,n_branch,n_prune)
461			end
462			
463			
464			#############
465			
466			"""
467			```
468			classicBP :: Function
469			```
470			This function is an important solver of this package. This solver implements the algorithm given in
471			
472			Liberti, L., Lavor, C., & Maculan, N. (2008). A branch-and-prune algorithm for the molecular distance geometry problem. International Transactions in Operational Research, 15(1), 1-17.
473			
474			The main difference of our implementation is that the input data (`NMRType`) can store informations of preprocessing function and as consequence we can optimize the search tree of this algorithm.
475			"""
476			function classicBP(NMRdata :: NMRType,
477			ε :: Float64,
478			virtual_ε :: Float64,
479			allmol :: Bool, time_limit)
480			
481			#start = Dates.now()
482			#time_elapsed = Second(0.0)
483			14650 (57.88%) samples spent in classicBP 1 (100.00%) (ex.), 14650 (100.00%) (incl.) when called from conformation line 42
484			n = NMRdata.dim
485			if n < 3
486			ArgumentError("Invalid dimension of NMRdata")
487			end
488			
489			virtual_ε² = virtual_ε*virtual_ε
490			nsol = 0
491			storage_mol = Dict{Int64,MoleculeType}()
492			
493			1 (0.0040%) samples spent in initialization 0 (ex.), 1 (100.00%) (incl.) when called from classicBP line 648 function initialization()
494			mol = MoleculeType(Vector{AtomType}(undef,n),0.0)
495			for i=1:n
496			mol.atoms[i] = AtomType(0.0,0.0,0.0,0.0)
497			end
498			C = zeros(4,4)
499			
500			# first atom
501			mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
502			mol.atoms[1].x = 0.0
503			mol.atoms[1].y = 0.0
504			mol.atoms[1].z = 0.0
505			#second atom
506		1 (0.0040%)	1 (100.00%) samples spent calling getindex mol.atoms[2].element = NMRdata.info[2,:].nzval[1].atom1
507			mol.atoms[2].x = -NMRdata.info[1,2].dist
508			mol.atoms[2].y = 0.0
509			mol.atoms[2].z = 0.0
510			# tird atom
511			D12 = NMRdata.info[1,2].dist
512			D13 = NMRdata.info[1,3].dist
513			D23 = NMRdata.info[2,3].dist
514			c0,s0 = bondangle(D12,D13,D23)
515			mol.atoms[3].element = NMRdata.info[3,:].nzval[1].atom1
516			mol.atoms[3].x = -D12+D23*c0
517			mol.atoms[3].y = D23*s0
518			mol.atoms[3].z = 0.0
519			C = zeros(4,4)
520			C[1,4] = mol.atoms[3].x
521			C[2,4] = mol.atoms[3].y
522			C[3,4] = mol.atoms[3].z
523			C[1,1] = c0
524			C[1,2] = s0
525			C[2,1] = s0
526			C[2,2] = -c0
527			C[3,3] = -1.0
528			C[4,4] = 1.0
529			
530			return 4,4,mol,C
531			end
532			
533			# defining closure
534			function classicBP_closure(l :: Int64,
535			pos::Int64,
536			mol :: MoleculeType,
537			C :: Array{Float64,2})
538			
539			#time_elapsed = Dates.now()-start
540			#if time_elapsed>time_limit && l<n
541			# error("Time limit reached without found a solution!")
542			#end
543			14648 (57.87%) samples spent in classicBP_closure 648 (50.78%) (ex.), 14619 (99.80%) (incl.) when called from classicBP_closure line 612 628 (49.22%) (ex.), 14630 (99.88%) (incl.) when called from classicBP_closure line 631 0 (ex.), 14648 (100.00%) (incl.) when called from classicBP line 641
544	15 (0.06%)	843 (3.33%)	828 (98.22%) samples spent calling copy C_before = copy(C)
545			while true
546	1 (0.0040%)	87 (0.34%)	1 (1.15%) samples spent calling getindex 8 (9.20%) samples spent calling getproperty 77 (88.51%) samples spent calling getindex virtualPos = NMRdata.virtual_path[pos]

547		1 (0.0040%)	1 (100.00%) samples spent calling getindex virtualLastPos = NMRdata.virtual_path[pos-1]
548			
549	1 (0.0040%)	7 (0.03%)	5 (71.43%) samples spent calling getindex 1 (14.29%) samples spent calling == if NMRdata.virtual_path[pos-3] == virtualPos
550			D14 = 0.0
551			else
552		193 (0.76%)	43 (22.28%) samples spent calling getindex 99 (51.30%) samples spent calling getindex 51 (26.42%) samples spent calling getproperty D14 = NMRdata.info[NMRdata.virtual_path[pos-3],virtualPos].dist
553			end
554	1 (0.0040%)	1 (0.0040%)	if NMRdata.virtual_path[pos-2] == virtualPos
555			D24 = 0.0
556			else
557		47 (0.19%)	6 (12.77%) samples spent calling getindex 19 (40.43%) samples spent calling getproperty 22 (46.81%) samples spent calling getindex D24 = NMRdata.info[NMRdata.virtual_path[pos-2],virtualPos].dist
558			end
559			
560			if virtualLastPos == virtualPos
561			D34 = 0.0
562			else
563		33 (0.13%)	17 (51.52%) samples spent calling getindex 5 (15.15%) samples spent calling getproperty 11 (33.33%) samples spent calling getindex D34 = NMRdata.info[virtualLastPos,virtualPos].dist
564			end
565			
566		3 (0.01%)	3 (100.00%) samples spent calling getindex if NMRdata.virtual_path[pos-3] == NMRdata.virtual_path[pos-2]
567			D12 = 0.0
568			else
569		89 (0.35%)	9 (10.11%) samples spent calling getindex 40 (44.94%) samples spent calling getindex 40 (44.94%) samples spent calling getproperty D12 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-2]].dist
570			end
571		3 (0.01%)	2 (66.67%) samples spent calling getindex 1 (33.33%) samples spent calling getproperty if NMRdata.virtual_path[pos-3] == virtualLastPos
572			D13 = 0.0
573			else
574		42 (0.17%)	25 (59.52%) samples spent calling getindex 14 (33.33%) samples spent calling getproperty 3 (7.14%) samples spent calling getindex D13 = NMRdata.info[NMRdata.virtual_path[pos-3],virtualLastPos].dist
575			end
576	1 (0.0040%)	6 (0.02%)	1 (16.67%) samples spent calling getproperty 4 (66.67%) samples spent calling getindex if NMRdata.virtual_path[pos-2] == virtualLastPos
577			D23 = 0.0
578			else
579		31 (0.12%)	4 (12.90%) samples spent calling getindex 13 (41.94%) samples spent calling getproperty 14 (45.16%) samples spent calling getindex D23 = NMRdata.info[NMRdata.virtual_path[pos-2],virtualLastPos].dist
580			end
581			
582		21 (0.08%)	5 (23.81%) samples spent calling bondangle 1 (4.76%) samples spent calling bondangle 2 (9.52%) samples spent calling bondangle 6 (28.57%) samples spent calling bondangle 5 (23.81%) samples spent calling bondangle 2 (9.52%) samples spent calling bondangle c0,s0 = bondangle(D23,D24,D34)
583		20 (0.08%)	1 (5.00%) samples spent calling badtorsionangle 10 (50.00%) samples spent calling badtorsionangle 1 (5.00%) samples spent calling badtorsionangle 8 (40.00%) samples spent calling badtorsionangle cw,sw = badtorsionangle(D12,D13,D14,D23,D24,D34)
584	283 (1.12%)	1404 (5.55%)	1121 (79.84%) samples spent calling torsionmatrix B = torsionmatrix(c0,s0,cw,sw,D34)
585			if l==virtualPos
586	4 (0.02%)	1042 (4.12%)	1038 (99.62%) samples spent calling prodmatrix C = prodmatrix(C_before,B)
587			break
588			else
589		51 (0.20%)	33 (64.71%) samples spent calling getproperty 18 (35.29%) samples spent calling getindex cpx = mol.atoms[virtualPos].x
590			cpy = mol.atoms[virtualPos].y
591		2 (0.0079%)	2 (100.00%) samples spent calling getproperty cpz = mol.atoms[virtualPos].z
592			
593	2 (0.0079%)	919 (3.63%)	1 (0.11%) samples spent calling prodmatrix 915 (99.56%) samples spent calling prodmatrix 1 (0.11%) samples spent calling prodmatrix Virtual_Torsion = prodmatrix(C_before,B)
594			
595		8 (0.03%)	2 (25.00%) samples spent calling >= 1 (12.50%) samples spent calling getindex 2 (25.00%) samples spent calling + 2 (25.00%) samples spent calling + 1 (12.50%) samples spent calling literal_pow if (Virtual_Torsion[1,4]- cpx)^2+(Virtual_Torsion[2,4]- cpy)^2+(Virtual_Torsion[3,4]- cpz)^2> virtual_ε²
596	2 (0.0079%)	59 (0.23%)	57 (96.61%) samples spent calling torsionmatrix B = torsionmatrix(B)
597		48 (0.19%)	48 (100.00%) samples spent calling prodmatrix C_before = prodmatrix(C_before,B)
598			else
599		108 (0.43%)	108 (100.00%) samples spent calling copy C_before = copy(Virtual_Torsion)
600			end
601			pos = pos+1
602			end
603			end
604			
605		6411 (25.33%)	6272 (97.83%) samples spent calling getindex 89 (1.39%) samples spent calling getproperty 12 (0.19%) samples spent calling setproperty! 38 (0.59%) samples spent calling getindex mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
606		56 (0.22%)	42 (75.00%) samples spent calling getindex 8 (14.29%) samples spent calling getindex 6 (10.71%) samples spent calling getproperty mol.atoms[1].x = C[1,4]
607		3 (0.01%)	3 (100.00%) samples spent calling getindex mol.atoms[1].y = C[2,4]
608		4 (0.02%)	1 (25.00%) samples spent calling getindex 2 (50.00%) samples spent calling setproperty! 1 (25.00%) samples spent calling getproperty mol.atoms[1].z = C[3,4]
609			
610	108 (0.43%)	1187 (4.69%)	1036 (87.28%) samples spent calling pruningtest 18 (1.52%) samples spent calling shiftleft!_4900.clone_1.clone 17 (1.43%) samples spent calling current_logger_for_env 5 (0.42%) samples spent calling getindex 3 (0.25%) samples spent calling pruningtest if pruningtest(mol,l,NMRdata,ε)
611	2 (0.0079%)	4 (0.02%)	2 (50.00%) samples spent calling < if l<n
612	238 (0.94%)	2717383 (10735.55%)	14647 (0.54%) samples spent calling classicBP_closure 7 (0.0003%) samples spent calling torsionmatrix 9 (0.0003%) samples spent calling pruningtest 7 (0.0003%) samples spent calling getindex 4 (0.0001%) samples spent calling getindex 1 samples spent calling prodmatrix classicBP_closure(l+1,pos+1,mol,C)
613			else
614	1 (0.0040%)	1 (0.0040%)	nsol=nsol+1
615			storage_mol[nsol] = copy(mol)
616	4 (0.02%)	4 (0.02%)	return
617			end
618			end
619	243 (0.96%)	282 (1.11%)	4 (1.42%) samples spent calling collect_similar_18398.clone 35 (12.41%) samples spent calling >_18399.clone if !allmol && nsol>0
620			return
621			end
622			
623	15 (0.06%)	309 (1.22%)	294 (95.15%) samples spent calling torsionmatrix B = torsionmatrix(B)

624	20 (0.08%)	255 (1.01%)	235 (92.16%) samples spent calling prodmatrix C = prodmatrix(C_before,B)# tenho que otimizar este calculo
625		39 (0.15%)	5 (12.82%) samples spent calling setproperty! 20 (51.28%) samples spent calling getindex 14 (35.90%) samples spent calling getproperty mol.atoms[1].x = C[1,4]
626		6 (0.02%)	3 (50.00%) samples spent calling setproperty! 1 (16.67%) samples spent calling getindex 2 (33.33%) samples spent calling getproperty mol.atoms[1].y = C[2,4]
627		4 (0.02%)	1 (25.00%) samples spent calling getproperty 3 (75.00%) samples spent calling getindex mol.atoms[1].z = C[3,4]
628			
629	18 (0.07%)	415 (1.64%)	388 (93.49%) samples spent calling pruningtest 1 (0.24%) samples spent calling shiftleft!_4980.clone_1.clone 4 (0.96%) samples spent calling current_logger_for_env 1 (0.24%) samples spent calling getindex 3 (0.72%) samples spent calling pruningtest if pruningtest(mol,l,NMRdata,ε) #preciso modificar
630	2 (0.0079%)	2 (0.0079%)	if l<n
631	290 (1.15%)	7347994 (29029.69%)	1 samples spent calling classicBP_closure 4 samples spent calling prodmatrix 14647 (0.20%) samples spent calling classicBP_closure 1 samples spent calling torsionmatrix 4 samples spent calling pruningtest 5 samples spent calling getindex 2 samples spent calling getindex classicBP_closure(l+1,pos+1,mol,C)
632			else
633			nsol = nsol+1
634	15 (0.06%)	15 (0.06%)	storage_mol[nsol] = copy(mol)
635			return
636			end
637			end
638			end # closure
639			
640		1 (0.0040%)	1 (100.00%) samples spent calling initialization __l, __pos, __mol, __C = initialization()
641		14648 (57.87%)	14648 (100.00%) samples spent calling classicBP_closure classicBP_closure(__l, __pos, __mol, __C)
642	1 (0.0040%)	1 (0.0040%)	return nsol, storage_mol
643			
644			end #solver classicBP
645			
646			"""
647			```
648			quaternionBP :: Function
649			```
650			This function defines a new solver. The implementation follows ideas describing in:
651			
652			Fidalgo, F. Using Quaternion Geometric Algebra for efficient rotations in Branch and Prune Algorithtm to solve the Discretizable Molecular Distance Geometry Problem. In: Proceedings of AGACSE 2018, Campinas-SP, Brazil.
653			"""
654			function quaternionBP(NMRdata :: NMRType,
655			ε :: Float64,
656			virtual_ε :: Float64,
657			allmol :: Bool, time_limit)
658			
659			#start = Dates.now()
660			#time_elapsed = Second(0.0)
661			10206 (40.32%) samples spent in quaternionBP 0 (ex.), 10206 (100.00%) (incl.) when called from conformation line 42
662			n = NMRdata.dim
663			if n < 3
664			ArgumentError("Invalid dimension of NMRdata")
665			end
666			
667			virtual_ε² = virtual_ε*virtual_ε
668			nsol = 0
669			storage_mol = Dict{Int64,MoleculeType}()
670			
671			function initialization()
672			# TODO: (Emerson) na criação desse vetor você não pode já estabelecer um valor default para atoms?
673			mol = MoleculeType(Vector{AtomType}(undef,n),0.0)
674			for i=1:n
675			mol.atoms[i] = AtomType(0.0,0.0,0.0)
676			end
677			Q = Quaternion(0.0,0.0,0.0,0.0)
678			
679			# first atom
680			mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
681			mol.atoms[1].x = 0.0
682			mol.atoms[1].y = 0.0
683			mol.atoms[1].z = 0.0
684			#second atom
685			mol.atoms[2].element = NMRdata.info[2,:].nzval[1].atom1
686			mol.atoms[2].x = -NMRdata.info[1,2].dist
687			mol.atoms[2].y = 0.0
688			mol.atoms[2].z = 0.0
689			# tird atom
690			D12 = NMRdata.info[1,2].dist
691			D13 = NMRdata.info[1,3].dist
692			D23 = NMRdata.info[2,3].dist
693			c0,s0 = qbondangle(D12,D13,D23)
694			Q = Quaternion(0.0,-c0,-s0,0.0)
695			d = 2.0*D23
696			qmol = Quaternion(0.0,d*(c0*c0-0.5),d*(c0*s0),0.0)
697			mol.atoms[3].element = NMRdata.info[3,:].nzval[1].atom1
698			mol.atoms[3].x = qmol.v1 + mol.atoms[2].x
699			mol.atoms[3].y = qmol.v2 + mol.atoms[2].y
700			mol.atoms[3].z = qmol.v3 + mol.atoms[2].z
701			
702			return 4,4,mol,Q
703			end
704			
705			# defining closure
706			function quaternionBP_closure(l :: Int64,
707			pos::Int64,
708			mol :: MoleculeType,
709			Q :: Quaternion)
710			
711			#time_elapsed = Dates.now()-start
712			#if time_elapsed>time_limit && l<n
713			# error("Time limit reached without found a solution!")
714			#end
715			10206 (40.32%) samples spent in quaternionBP_closure 460 (52.69%) (ex.), 10195 (99.89%) (incl.) when called from quaternionBP_closure line 813 413 (47.31%) (ex.), 10196 (99.90%) (incl.) when called from quaternionBP_closure line 795 0 (ex.), 10206 (100.00%) (incl.) when called from quaternionBP line 823
716	13 (0.05%)	13 (0.05%)	lastpos = 1
717			D34 = 0.0
718			virtuallastPos = 0.0
719			a = 0.0
720			b = 0.0
721			c = 0.0
722			d = 0.0
723		20 (0.08%)	20 (100.00%) samples spent calling copy Q_before = copy(Q)
724			
725			while true
726		77 (0.30%)	3 (3.90%) samples spent calling getindex 5 (6.49%) samples spent calling getproperty 69 (89.61%) samples spent calling getindex virtualPos = NMRdata.virtual_path[pos]
727		4 (0.02%)	4 (100.00%) samples spent calling getindex virtualLastPos = NMRdata.virtual_path[pos-1]
728			
729		3 (0.01%)	1 (33.33%) samples spent calling == 2 (66.67%) samples spent calling getindex if NMRdata.virtual_path[pos-3] == virtualPos
730			D14 = 0.0
731			else
732		167 (0.66%)	25 (14.97%) samples spent calling getindex 47 (28.14%) samples spent calling getproperty 95 (56.89%) samples spent calling getindex D14 = NMRdata.info[NMRdata.virtual_path[pos-3],virtualPos].dist

733			end
734			if NMRdata.virtual_path[pos-2] == virtualPos
735			D24 = 0.0
736			else
737		39 (0.15%)	7 (17.95%) samples spent calling getindex 19 (48.72%) samples spent calling getindex 13 (33.33%) samples spent calling getproperty D24 = NMRdata.info[NMRdata.virtual_path[pos-2],virtualPos].dist
738			end
739			
740			if virtualLastPos == virtualPos
741			D34 = 0.0
742			else
743		42 (0.17%)	8 (19.05%) samples spent calling getindex 20 (47.62%) samples spent calling getindex 14 (33.33%) samples spent calling getproperty D34 = NMRdata.info[virtualLastPos,virtualPos].dist
744			end
745			
746		4 (0.02%)	4 (100.00%) samples spent calling getindex if NMRdata.virtual_path[pos-3] == NMRdata.virtual_path[pos-2]
747			D12 = 0.0
748			else
749		83 (0.33%)	30 (36.14%) samples spent calling getindex 41 (49.40%) samples spent calling getproperty 12 (14.46%) samples spent calling getindex D12 = NMRdata.info[NMRdata.virtual_path[pos-3],NMRdata.virtual_path[pos-2]].dist
750			end
751	1 (0.0040%)	4 (0.02%)	2 (50.00%) samples spent calling getindex 1 (25.00%) samples spent calling == if NMRdata.virtual_path[pos-3] == virtualLastPos
752			D13 = 0.0
753			else
754		51 (0.20%)	28 (54.90%) samples spent calling getindex 16 (31.37%) samples spent calling getproperty 7 (13.73%) samples spent calling getindex D13 = NMRdata.info[NMRdata.virtual_path[pos-3],virtualLastPos].dist
755			end
756	1 (0.0040%)	3 (0.01%)	2 (66.67%) samples spent calling getindex if NMRdata.virtual_path[pos-2] == virtualLastPos
757			D23 = 0.0
758			else
759	1 (0.0040%)	36 (0.14%)	4 (11.11%) samples spent calling getindex 4 (11.11%) samples spent calling getproperty 27 (75.00%) samples spent calling getindex D23 = NMRdata.info[NMRdata.virtual_path[pos-2],virtualLastPos].dist
760			end
761			
762		19 (0.08%)	13 (68.42%) samples spent calling qbondangle 1 (5.26%) samples spent calling qbondangle 5 (26.32%) samples spent calling qbondangle c0,s0 = qbondangle(D23,D24,D34)
763	3 (0.01%)	29 (0.11%)	26 (89.66%) samples spent calling qtorsionangle cw,sw = qtorsionangle(D12,D13,D14,D23,D24,D34)
764			a = s0*cw
765		5 (0.02%)	5 (100.00%) samples spent calling * b = s0*sw
766		2 (0.0079%)	2 (100.00%) samples spent calling * c = -c0*sw
767			d = c0*cw
768			if l==virtualPos
769			Q = qprod(Q_before,a,b,c,d)
770			lastpos = pos
771			break
772			else
773		897 (3.54%)	898 (99.22%) samples spent calling qprod 6 (0.67%) samples spent calling qprod 1 (0.11%) samples spent calling qprod Q_virtual = qprod(Q_before,a,b,c,d)
774		2 (0.0079%)	2 (100.00%) samples spent calling rotopt qmol = rotopt(Q_virtual,D34)
775			# TODO: (Emerson) não conseguimos fazer o calculo abaixo da mesma forma que o classicBP?
776		13 (0.05%)	10 (76.92%) samples spent calling getindex 1 (7.69%) samples spent calling + 2 (15.38%) samples spent calling - vx = qmol.v1 + mol.atoms[virtualLastPos].x-mol.atoms[virtualPos].x
777		4 (0.02%)	2 (50.00%) samples spent calling + 2 (50.00%) samples spent calling - vy = qmol.v2 + mol.atoms[virtualLastPos].y-mol.atoms[virtualPos].y
778			vz = qmol.v3 + mol.atoms[virtualLastPos].z-mol.atoms[virtualPos].z
779		3 (0.01%)	2 (66.67%) samples spent calling + 1 (33.33%) samples spent calling * if vx*vx + vy*vy + vz*vz > virtual_r²
780		4 (0.02%)	4 (100.00%) samples spent calling qprod Q_before = qprod(Q_before,a,-b,-c,d)
781			else
782			Q_before = Q_virtual
783			end
784			pos = pos+1
785			end
786			end
787		4 (0.02%)	4 (100.00%) samples spent calling rotopt qmol = rotopt(Q,D34)
788		6302 (24.90%)	68 (1.08%) samples spent calling getproperty 6288 (98.51%) samples spent calling getindex 8 (0.13%) samples spent calling setproperty! 18 (0.29%) samples spent calling getindex mol.atoms[1].element = NMRdata.info[1,:].nzval[1].atom1
789		50 (0.20%)	6 (12.00%) samples spent calling getproperty 10 (20.00%) samples spent calling + 34 (68.00%) samples spent calling getindex mol.atoms[1].x = qmol.v1 + mol.atoms[virtualLastPos].x
790			mol.atoms[1].y = qmol.v2 + mol.atoms[virtualLastPos].y
791		3 (0.01%)	2 (66.67%) samples spent calling + 1 (33.33%) samples spent calling getindex mol.atoms[1].z = qmol.v3 + mol.atoms[virtualLastPos].z
792			
793	97 (0.38%)	1140 (4.50%)	1 (0.09%) samples spent calling pruningtest 1004 (88.07%) samples spent calling pruningtest 2 (0.18%) samples spent calling getindex 17 (1.49%) samples spent calling shiftleft!_4900.clone_1.clone 19 (1.67%) samples spent calling current_logger_for_env if pruningtest(mol,1,NMRdata,ε)
794			if l<n
795	266 (1.05%)	1786389 (7057.48%)	1 samples spent calling quaternionBP_closure 1 samples spent calling qtorsionangle 1 samples spent calling quaternionBP_closure 2 (0.0001%) samples spent calling getindex 1 samples spent calling getindex 4 (0.0002%) samples spent calling pruningtest 10206 (0.57%) samples spent calling quaternionBP_closure quaternionBP_closure(l+1,pos+1,mol,Q)
796			else
797	1 (0.0040%)	1 (0.0040%)	nsol=nsol+1
798		19 (0.08%)	16 (84.21%) samples spent calling copy 3 (15.79%) samples spent calling copy storage_mol[nsol] = copy(mol)
799	13 (0.05%)	13 (0.05%)	return
800			end
801			end
802	216 (0.85%)	250 (0.99%)	34 (13.60%) samples spent calling >_18399.clone if !allmol && nsol>0
803			return
804			end
805		41 (0.16%)	1 (2.44%) samples spent calling qprod 40 (97.56%) samples spent calling qprod Q = qprod(Q_before,a,-b,-c,d)
806		9 (0.04%)	9 (100.00%) samples spent calling rotopt qmol = rotopt(Q,D34)
807		30 (0.12%)	13 (43.33%) samples spent calling getindex 1 (3.33%) samples spent calling setproperty! 13 (43.33%) samples spent calling + 3 (10.00%) samples spent calling getproperty mol.atoms[1].x = qmol.v1 + mol.atoms[virtualLastPos].x
808			mol.atoms[1].y = qmol.v2 + mol.atoms[virtualLastPos].y
809		4 (0.02%)	3 (75.00%) samples spent calling + 1 (25.00%) samples spent calling setproperty! mol.atoms[1].z = qmol.v3 + mol.atoms[virtualLastPos].z
810			
811	10 (0.04%)	278 (1.10%)	3 (1.08%) samples spent calling pruningtest 261 (93.88%) samples spent calling pruningtest 2 (0.72%) samples spent calling shiftleft!_4900.clone_1.clone 2 (0.72%) samples spent calling current_logger_for_env if pruningtest(mol,1,NMRdata,ε)
812			if l<n

813	241 (0.95%)	4853566 (19174.96%)	4 samples spent calling gruningtest 10206 (8.21%) samples spent calling quaternionBP_closure 3 samples spent calling getindex 3 samples spent calling getindex 1 samples spent calling + quaternionBP_closure(l+1,pos+1,mol,Q)
814			else
815			nsol = nsol+1
816	9 (0.04%)	9 (0.04%)	storage_mol[nsol] = copy(mol)
817			return
818			end
819			end
820			end #closure
821			
822			__l, __pos, __mol, __Q = initialization()
823		10206 (40.32%)	10206 (100.00%) samples spent calling quaternionBP_closure quaternionBP_closure(__l, __pos, __mol, __Q)
824			
825			return nsol, storage_mol
826			
827			end #solver quaternionBP
828			