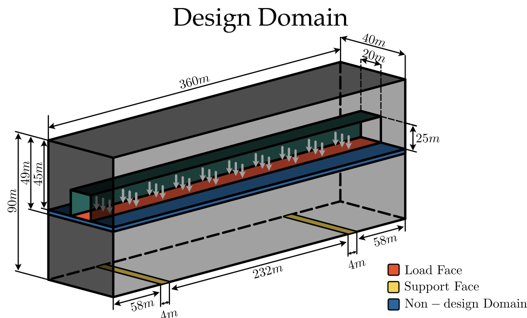
Run-time from 300 years to 300 min: Lessons learned in large-scale modeling in FEniCS.

Abhinav Gupta, U Meenu Krishnan, Rajib Chowdhury, Anupam Chakrabarti

Department of Civil Engineering
Indian Institute of Technology Roorkee, India

23 March 2021

Motivation



(I) Solve the topology optimization problem for a medium to large scale engineering structure.

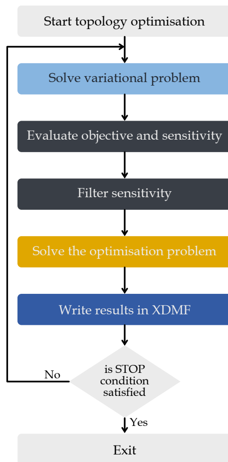
(II) The problem could contain degrees of freedom ranging from a million to over a billion.

Coding topology optimization in FEniCS

$$\underset{\theta}{\text{minimize}} \quad f(\theta) = \theta^p \psi(\boldsymbol{\epsilon})$$

$$\text{subject to} \quad g(\theta) = \int_\Omega \theta \, \mathrm{d}\Omega - kV_0 \leq 0$$

$$\psi(\boldsymbol{\epsilon}) = E_0 \left[\frac{\lambda_0}{2} (tr[\boldsymbol{\epsilon}])^2 + \mu_0 tr[\boldsymbol{\epsilon}^2] \right]$$



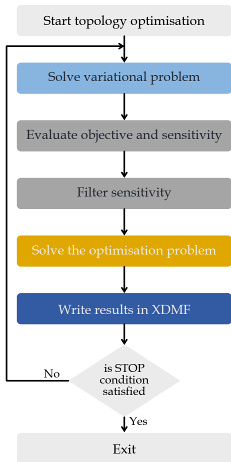

```matlab
function top(nelx,nely,volfrac,penal,rmin);
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk;
  c = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
       ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end
```

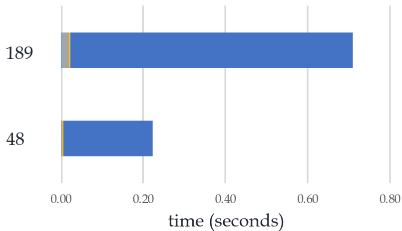
MATLAB
99-Line code

```python
while change > 0.01 and loop < 100:
    loop = loop + 1
    density_old.assign(density)
    # FE-ANALYSIS ---------------------------------------------------------
    solver.solve()
    # OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS ------------------
    objective = density ** penal * psi(u_sol)
    sensitivity = project(-diff(objective, density), D).vector()[:]
    # FILTERING/MODIFICATION OF SENSITIVITIES ---------------------
    sensitivity = np.divide(distance_mat @ np.multiply(density.vector()[:
        ], sensitivity), np.multiply(density.vector()[:], distance_sum))
    # DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD -------------
    l1, l2, move = 0, 100000, 0.2
    while l2 - l1 > 1e-4:
        l_mid = 0.5 * (l2 + l1)
        density_new.vector()[:] = np.maximum(0.001, np.maximum(density.
            vector()[:] - move, np.minimum(1.0, np.minimum(density.vector
            ()[:] + move, density.vector()[:] * np.sqrt(-sensitivity / V0
             / l_mid)))))
        current_vol = assemble(density_new * dx)
        l1, l2 = (l_mid, l2) if current_vol > volfrac * V0.sum() else (l1
            , l_mid)
    # PRINT RESULTS -------------------------------------------------------
    change = max(density_new.vector()[:] - density_old.vector()[:])
    print("it.: {0} , obj.: {1:.3f} Vol.: {2:.3f}, ch.: {3:.3f}".format(
        loop, project(objective, D).vector().sum(), current_vol / V0.sum(
        ), change))
    density.assign(density_new)
    xdmf.write(density, loop)
```

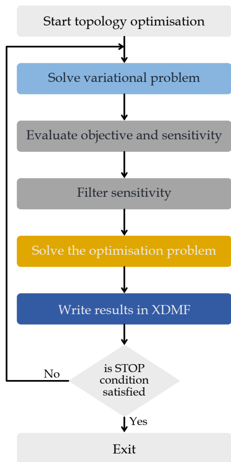
FEniCS
55-Line code

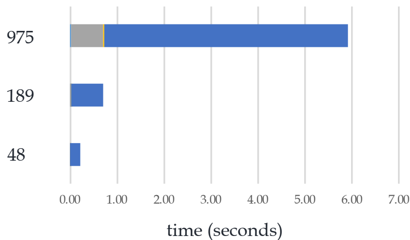
The problem



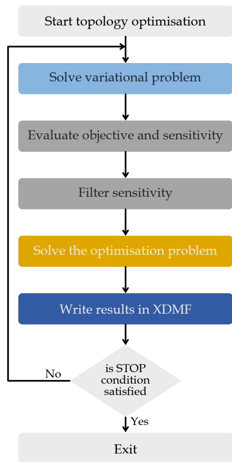


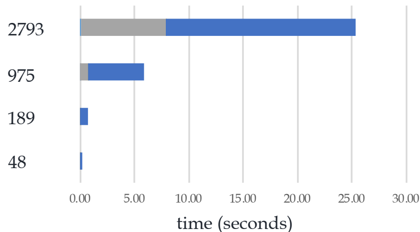
The problem



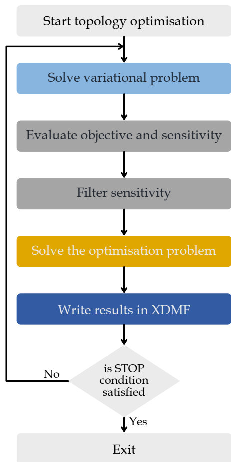


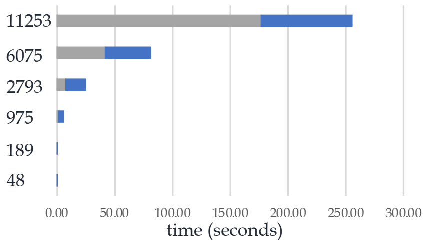
The problem



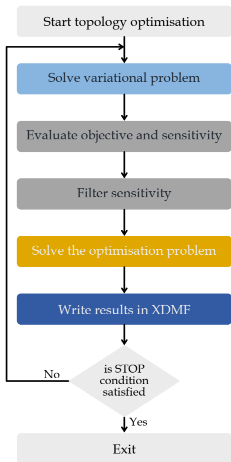


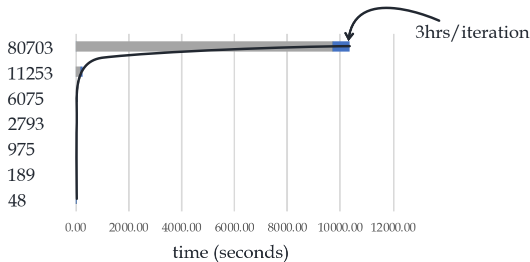
The problem



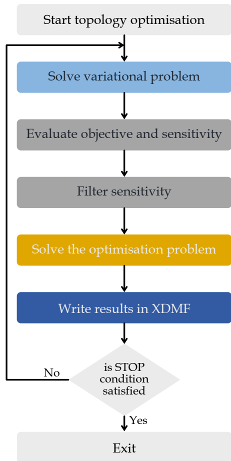


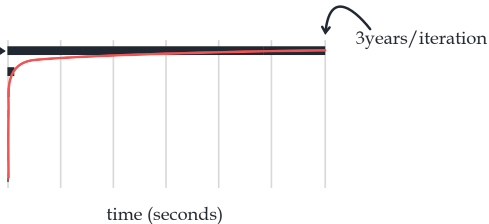
The problem

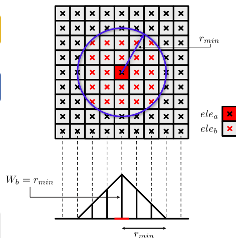




Loops are excruciatingly slow.








```
# PREPARE DISTANCE MATRICES FOR FILTER ---------------------------
midpoint = [cell.midpoint() for cell in cells(mesh)]
row_ind, col_ind, data = [], [], []
for row, ele_a in enumerate(midpoint):
    for col, ele_b in enumerate(midpoint):
        distance = ele_a.distance(ele_b)
        if distance < rmin + pow(10.0,-12.0):
            row_ind.append(row)
            col_ind.append(col)
            data.append(distance)
distance_mat = sp.csr_matrix((data, (row_ind, col_ind)), shape = (len(
    midpoint), len(midpoint)))
distance_sum = np.array(distance_mat.sum(1))[:, 0]
```

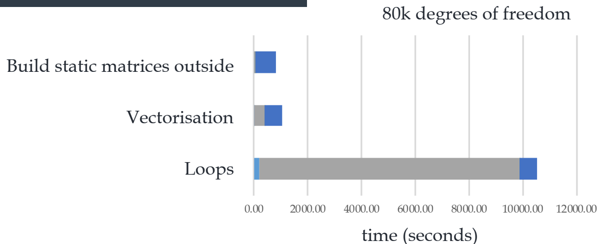
Vectorization and static matrices

Loops

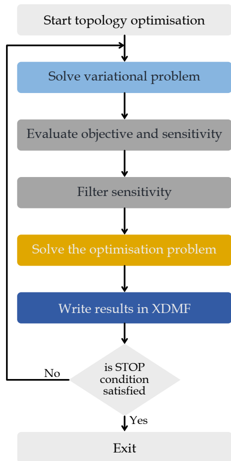
```
# PREPARE DISTANCE MATRICES FOR FILTER -----------------------------
    midpoint = [cell.midpoint() for cell in cells(mesh)]
    row_ind, col_ind, data = [], [], []
    for row, ele_a in enumerate(midpoint):
        for col, ele_b in enumerate(midpoint):
            distance = ele_a.distance(ele_b)
            if distance < rmin + pow(10.0,-12.0):
                row_ind.append(row)
                col_ind.append(col)
                data.append(distance)
    distance_mat = sp.csr_matrix((data, (row_ind, col_ind)), shape = (len(
        midpoint), len(midpoint)))
    distance_sum = np.array(distance_mat.sum(1))[:, 0]
```

Vectorisation

```
# PREPARE DISTANCE MATRICES FOR FILTER -----------------------------
midpoint = [cell.midpoint().array()[:] for cell in cells(mesh)]
distance_mat = np.maximum(rmin - sp.euclidean_distances(midpoint,
    midpoint), 0)
distance_sum = distance_mat.sum(1)
```



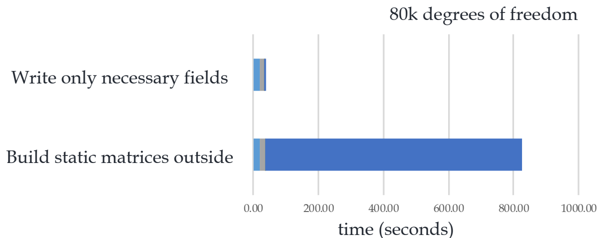
Controlled post processing



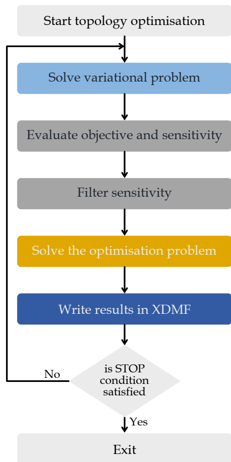

```
xdmf.write(project(psi(u_sol),D), loop)
```

Each call to project results in a call to solve for approximating the field by finite element method.

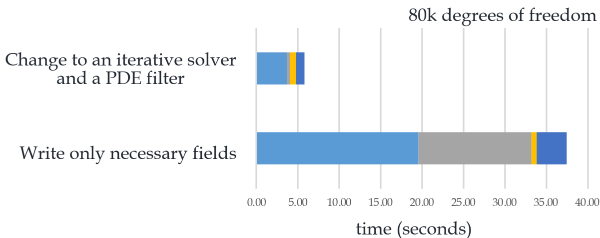
$$w = E_0 \left[\frac{\lambda_0}{2} (tr[\epsilon])^2 + \mu_0 tr[\epsilon^2] \right]$$



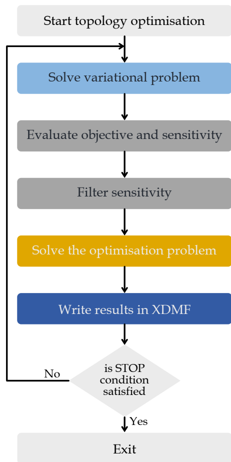
Properly select/configure the solver and preconditioner.

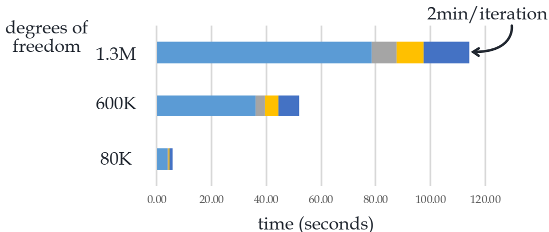


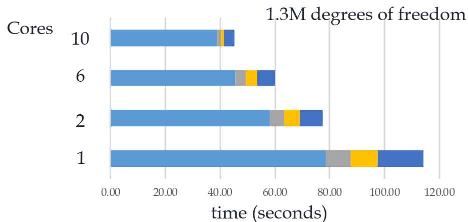

```python
problem = LinearVariationalProblem(K, F, u_sol, bcs)
solver = LinearVariationalSolver(problem)
solver.parameters["linear_solver"] = "gmres"
solver.parameters["preconditioner"] = "hypre_euclid"

project(psi(u_sol),D, solver_type="gmres", preconditioner_type="hypre_euclid")
```



Parallelization.

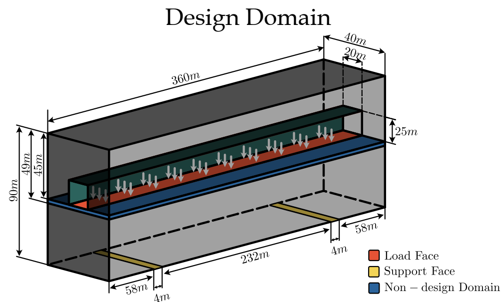






Result

Design Domain



Topologically optimised design

Conclusion

(I) General guidelines for handling medium to large-scale systems in FEniCS
  (i) Always profile the code and look for bottlenecks.
  (ii) Avoid use of loops in python. Look for efficient alternatives.
  (iii) Avoid re-evaluation of matrices that do not change.
  (iv) Evaluate and write only necessary simulation outputs.
  (v) In an iterative process evaluate output at every n^{th} step to further speed up the simulation.
  (vi) Properly select/configure the solver and preconditioner based on the problem.

(II) Stepping into the realm of large scale simulations require knowledge of good programming practices, parallelization, and a deep understanding of the working principles of the tools/libraries.

Thanks

---

iitrabhi@gmail.com
computationalmechanics.in