parWalLS: A Fast Parallel implementation of WalLS using Cython

Wenxiang Chen
Email: chenwx@cs.colostate.edu, URL: http://www.cs.colostate.edu/~chenwx
SCHEDuling Group, Computer Science Department, Colorado State University

Colorado State University
Knowledge to Go Places

Motivation

Pseudo-Boolean function is function that maps $\{0,1\}^n$ to R.

$$f(\boldsymbol{x}) = a + \sum_i a_i x_i + \sum_{i<j} a_{ij} x_i x_j + \sum_{i<j<k} a_{ijk} x_j x_j x_k + \dots$$

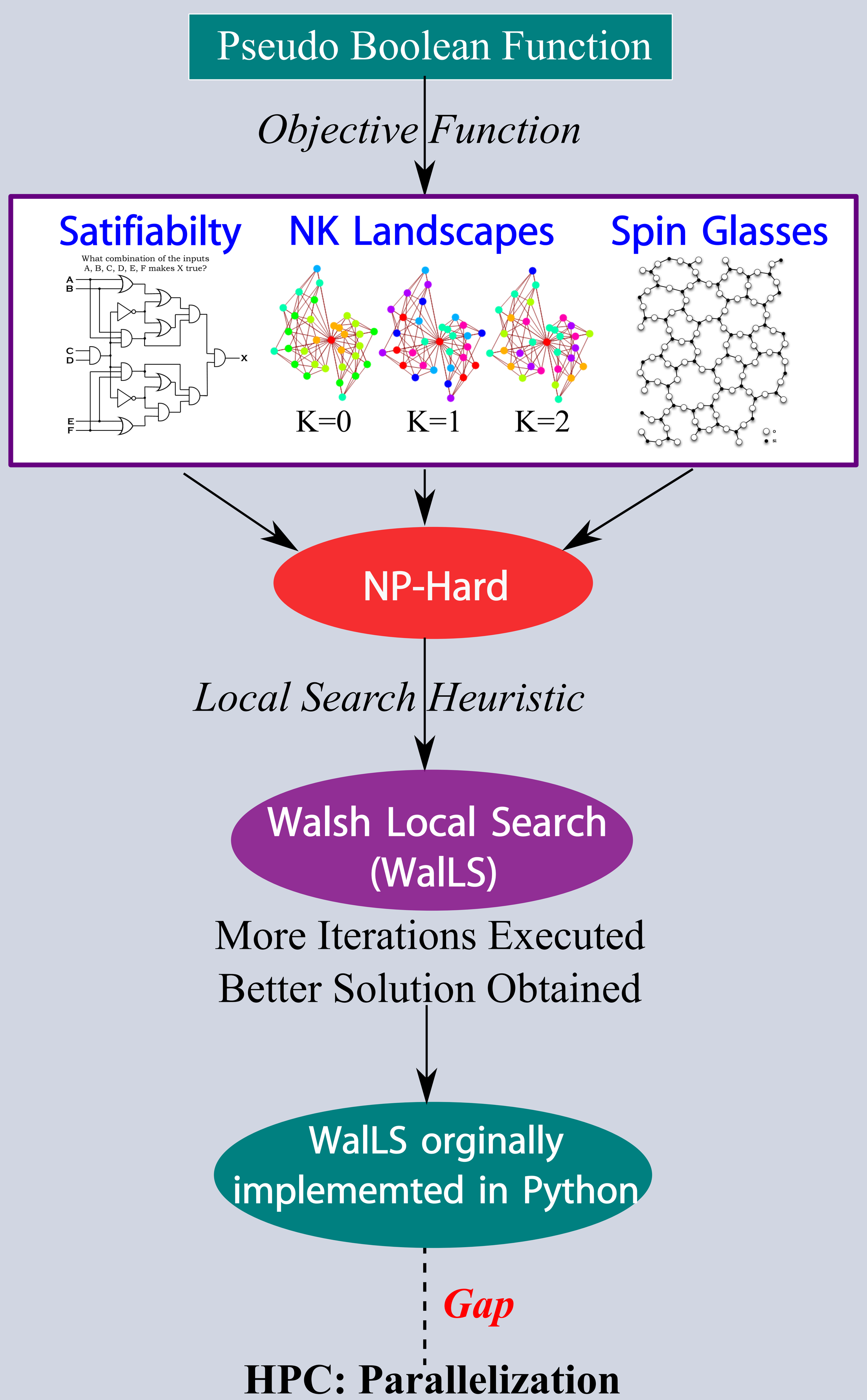


Figure: Motivation flow

- Pseudo-Boolean functions act as the objective functions in a variety of fundamental and well-studied combinatorial optimization problems from various research domains.
- e.g, Maximum-Satisfiability (Max-Sat) from Computer Science, NK-Landscape from Theoretic Biology and Spin Glasses Model from Physics.
- These optimization problems have been proven to be NP-Hard. No algorithm has been found for solving NP problems in polynomial time.
- Randomized Heuristics, such as Stochastic Local Search (SLS). SLS can discover solution of good quality in a reasonable amount of time.
- For SLS, more iteration executed leads to better solutions obtained. We recently develop a algorithmic efficient algorithm, "Constant Time Walsh Local Search based on Walsh Analysis (WalLS)" (GECCO'2012, PPSN'2012). WalLS takes O(1) cost for performing one steepest descent move, compared with $O(N)$ in conventional approach.
- However, WalLS is originally implemented in Python, which makes it inefficient in execution and infeasible to parallelize. There is still huge room to improve practical efficiency of WalLS.
- In this study, we aim at accelerating WalLS by leveraging between the convenience of High level programming language Python and execution efficiency of low-level language C, and introducing parallelization intro WalLS to speedup WalLS furthermore.

Our Approach

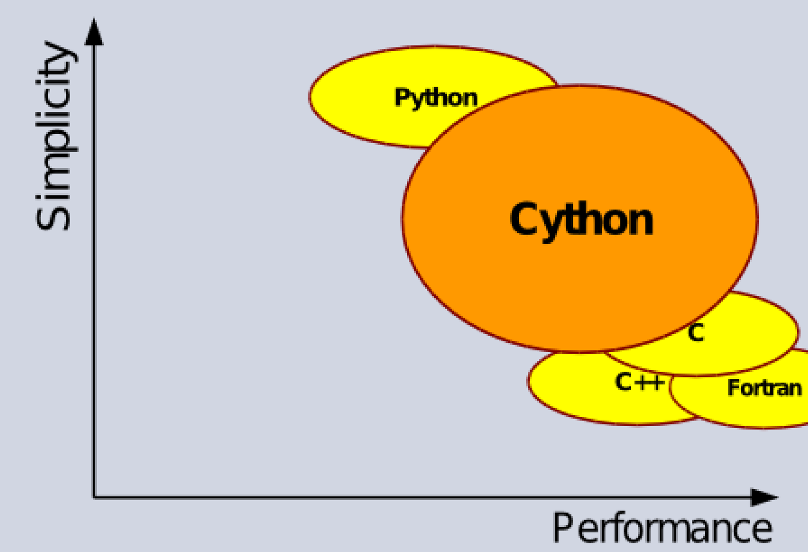


Figure: Simplicity Versus Performance

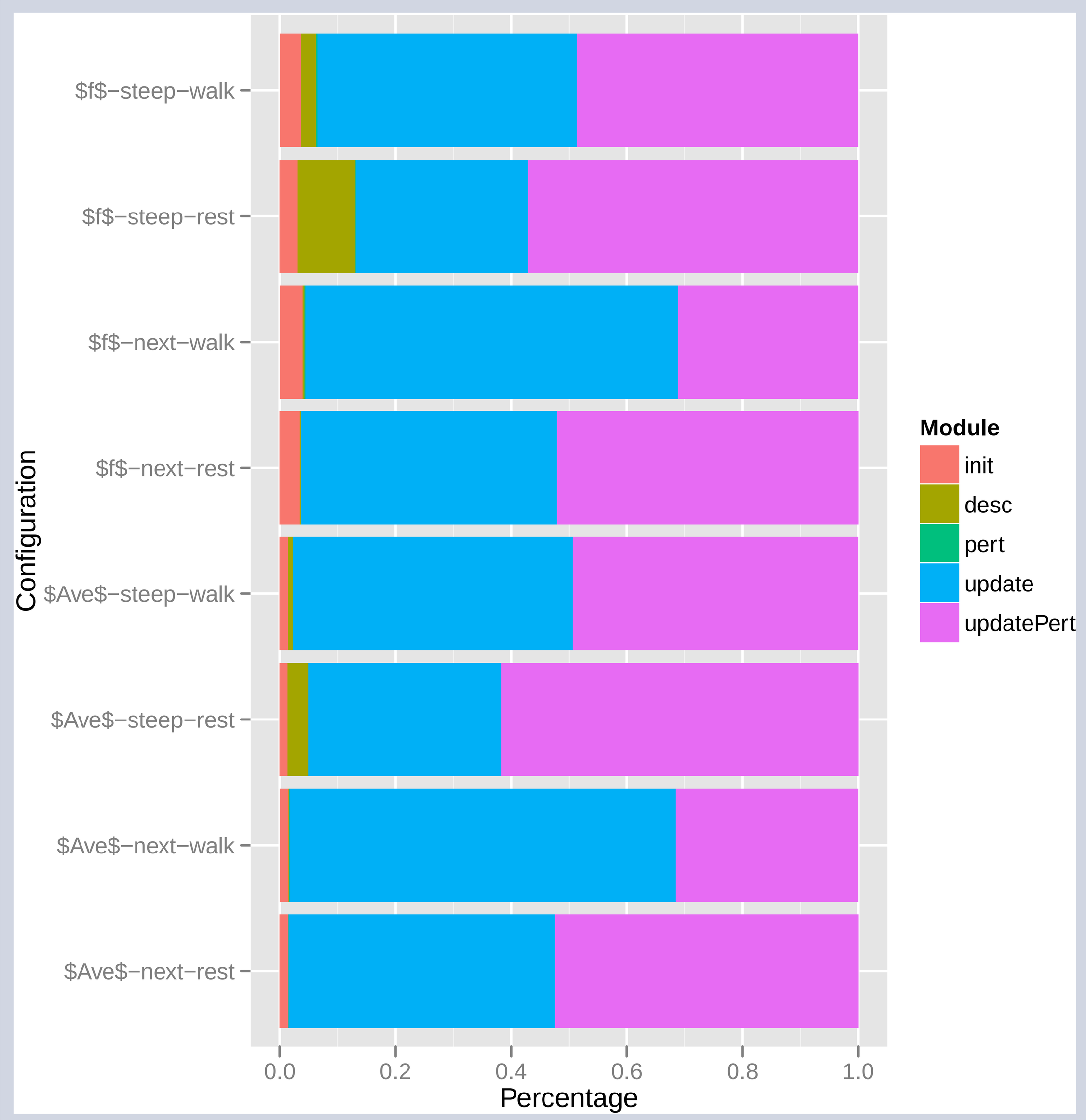
- Cython allows we to fill in the Gap between Python and HPC.
- achieves up to $41\times$ speedup without parallelization!
- even more acceleration with parallelization.

Sketch out of WalLS

Algorithm 1: Sol←WALLS(eval,descMeth,perturb)

bsfSol ← curSol ← INIT() s, z, buffer ← WALSHANALYSIS(f, curSol) ;
while *Termination criterion not met* **do**
  improve, bestI ← DESCENT(buffer, descMeth) ;
  if *improve == True* **then**
    w, s, z, buffer ← UPDATE(w, s, z, buffer, bestI, eval);
    curSol ← FLIP(curSol, bestI)
  else // local optimum: perturbation
    bsfSol ← SELECT(curSol, bsfSol) curSol ← PERTURBATION(curSol, perturb) ;
    for *i* **in** *DifferentBit(bsfSol, curSol)* **do**
      w, s, z, buffer ← UPDATE(w, s, z, buffer i, eval);
bsfSol ← SELECT(curSol, bsfSol) ;
return *bsfSol*

Performance bottleneck of WaLS



- UPDATE module (**update** + **updatePert**) clearly dominates the runtime ($\geq 90\%$)
- optimization effort should be devoted to UPDATE module

Why is Update module expensive?

Algorithm 2: w, s, z, buffer ← UPDATE(w, s, z, buffer, p, eval)

1 s[p] ← -s[p];
2 **for** *each q interacts with p* **do**
3   **for** *each w[i] touching both p and q* **do**
4   s[q] ← s[q] - 2 * w[i];
5 **for** *each s[i] touching p* **do**
6   **if** *s[i] is an improving move* **then**
7   buffer ← APPEND(buffer, i);
8 **for** *each w[i] touching p* **do**
9   w[i] ← -w[i]

- **Theorem**: UPDATE function in refalg:update takes $a_1 K 2^K + a_2 2^K + a_3$ time.
- Doubly nested loop at Line 2 to Line 4 accounts for the $a_1 K 2^K$ part.

How do we deal with it?

- conduct incremental optimization using Cython, in an attempt to generate highly efficient serial code (**Succeed**)
- introduce parallelization to better utilize multi-core computer (**Succeed**)
- automatic parallelization using POCC (*Fail*)

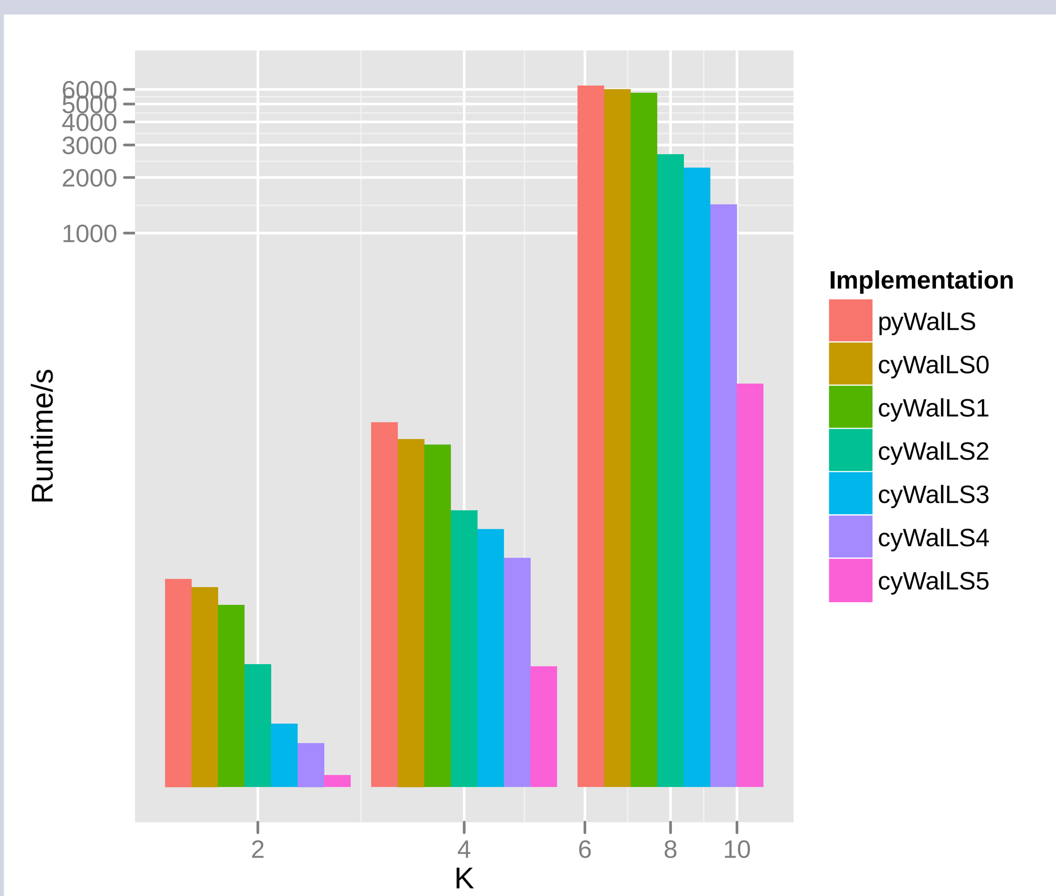
Why is Python slow?

- its interpreter → Cython is *complied*
- dictionary lookup → Cython has *cdef attributes*
- complicated calling conventions → Cython has *cdef functions*
- object-oriented primitives → Cython has *cdef values and types*

What does Cython do?

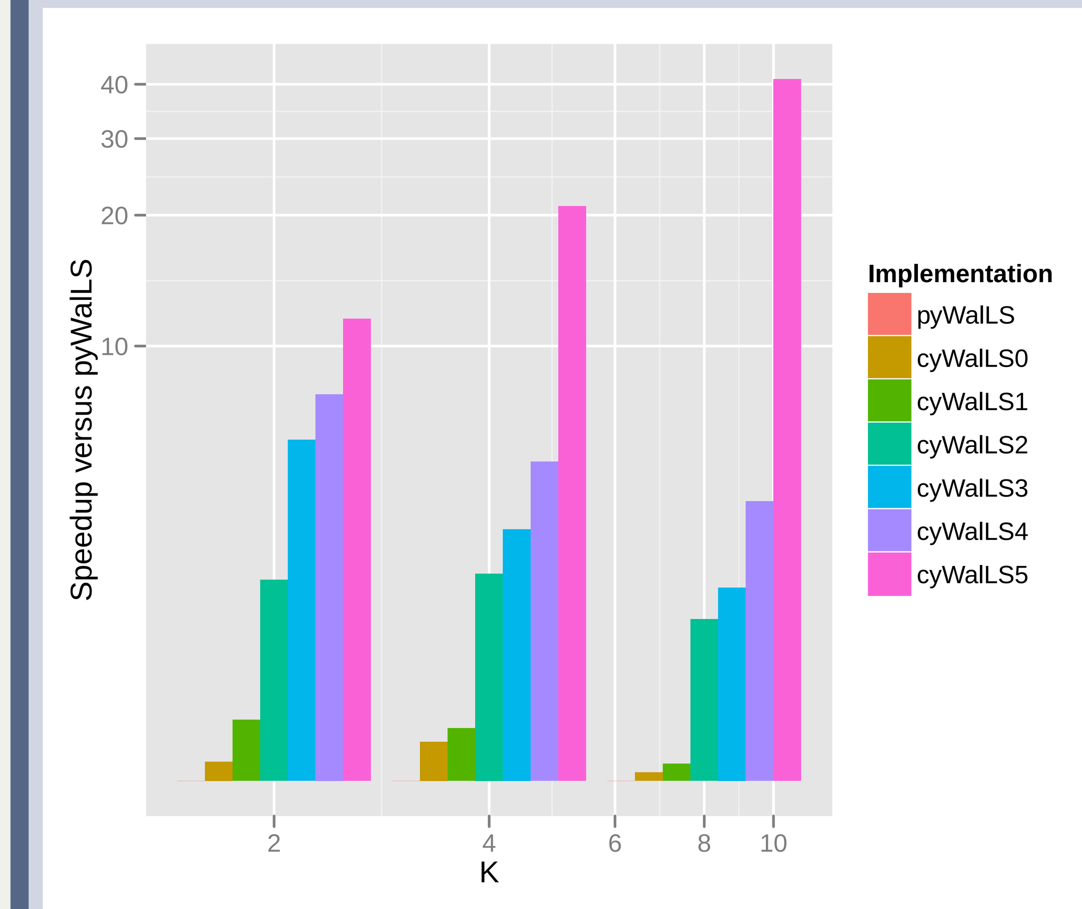
- compile Python into machine-code libraries, which can be imported from Python
- typing variables/functions
- ability to interfere with C/C++

Runtime Comparison



- as more optimizations are added, serial code gets faster and faster
- the later the version is, more sophisticated optimizations were used, the more speedup can be achieved
- simple optimizations include typing functions and variables
- sophisticated optimization contain rewriting Python object datatype using Struct/Class in C/C++

Speedup



As much as $41\times$ speedup can be achieved.

Parallelizing Kernel

Listing 1: Parallelized updateDeep function

```
cdef void updateDeep(self, int p):
  cdef int i, k0, k1
  cdef vector[int].iterator it
  cdef vector[int] arr
  cdef InfBit I
  # update the rest of elements in C matrix
  if self.infectBit[p].size() != 0:
    for i in prange(self.infectBit[p].size(), nogil=True
      ):
      I = self.infectBit[p][0][i]
      arr = I.arr[0]
      it = arr.begin()
      while it != arr.end():
        if deref(it) == p:
          arr.erase(it)
          break
        inc(it)
      comb = self.genComb(arr.size())
      for k in xrange(comb.size):
        k0 = arr[int(comb.arr[k][0])]
        k1 = arr[int(comb.arr[k][1])]
        self.C[k0][k1] = self.C[k0][k1] - 2 * self.WAS[I
          .WI].w
```

#Threads	1	2	Speedup
T/s	22.79	42.57	1.86

Table: Runtime of UPDATEDEEP function

Issues

- discover two bugs in Cython
  - one has been fixed in Cython 0.16, yet CS machs only provide v0.15.1.
  - the other has been confirmed by Cython core developers
- scalability evaluation is limited due to my dual-core laptop
- stochastic nature of WalLS fails the data-flow analysis → can not perform automatic transformation
  - Parameter p is not determined until runtime. All loops in Algorithm 2 rely on p, which make it implausible to analyze data-flow. Automation tools based-on data-flow analysis and polyhedron model fail to perform this kernel.

Conclusions

- achieve impressive speedup as much as $41\times$
- initialize study towards parallelization of WalLS
- evaluate the usability of Cython
- identify certain limitations of Cython
- pose a new chanllenge