Introduction to COIN-OR: Open Source Software for Optimization

TED RALPHS
ISE Department
COR@L Lab
Lehigh University
ted@lehigh.edu

University of Canterbury, 20 February 2009

Thanks: Work supported in part by the National Science Foundation

Outline

The Genesis

- An increasing number of papers being written in OR today are computational in nature or have a computational component.
- Historically, the pace of computational research has been relatively slow and the transfer of knowledge to practitioners has been even slower.
- Results of research are generally not reproducible.
- Research codes are buggy, narrowly focused, and lack robustness.
- There are few rewards for publishing software outside of archival journals.
- There is no peer review process for software and referees of computational papers have little to go on.
- Building on previous results is difficult and time-consuming.
- Interoperating with other software libraries (such as LP solvers) is difficult.
- The paradigm encouraged by archival journals does not work well for computational research.

The COIN-OR Initiative

- To address some of these challenges, the Common Optimization Interface for Operations Research Initiative was launched by IBM at ISMP in 2000.
- IBM seeded the repository with four initial projects, has hosted its Web site, and has provided funding.
- The goal was to develop the project and then hand it over to the community.
- The project has now grown to be self-sustaining and was spun off as a nonprofit educational foundation in the U.S. a few years ago.
- The name was also changed to the Computational Infrastructure for Operations Research to reflect a broader mission.

What is COIN-OR?

- The COIN-OR Foundation
  - A non-profit foundation promoting the development and use of interoperable, open-source software for operations research.
  - A consortium of researchers in both industry and academia dedicated to improving the state of computational research in OR.
  - A venue for developing and maintaining standards.
  - A forum for discussion and interaction between practitioners and researchers.
- The COIN-OR Repository
  - A library of interoperable software tools for building optimization codes, as well as a few stand alone packages.
  - A venue for peer review of OR software tools.
  - A development platform for open source projects, including an SVN repository,
- See `www.coin-or.org` for more information.

Our Agenda

- Accelerate the pace of research in computational OR.
  - Reuse instead of reinvent.
  - Reduce development time and increase robustness.
  - Increase interoperability (standards and interfaces).
- Provide for software what the open literature provides for theory.
  - Peer review of software.
  - Free distribution of ideas.
  - Adherence to the principles of good scientific research.
- Define standards and interfaces that allow software components to interoperate.
- Increase synergy between various development projects.
- Provide robust, open-source tools for practitioners.

Current Status

- The foundation has been up and running for several years and we are growing quickly!
- We currently have 30+ projects and a number in the queue.
- The foundation is run by two boards.
  - A strategic board to set overall direction
  - A technical board to advise on technical issues
- The boards are composed of members from both industry and academia, as well as balanced across disciplines.
- Membership in the foundation is available to both individuals and institutions.
- The foundation Web site and repository is hosted by INFORMS.

What is Open Source?

- A coding paradigm in which development is done in a cooperative and distributed fashion.
- An economic model used by some “for-profit” software ventures.
- This model is followed by a number of well-known software projects.
  - Linux (Red Hat, etc.)
  - Netscape/Mozilla (AOL)
  - Star Office/Open Office (Sun)
  - Apache
- A type of software license (described on the next slide).
- To find out more, see `www.opensource.org` or the writings of Eric S. Raymond (*The Cathedral and the Bazaar*).

Open Source Licenses

- Strictly speaking, an open source license must satisfy the requirements of the *Open Source Definition*.
- A license can/should not call itself “open source” until it is approved by the Open Source Initiative.
- Basic properties of an open source license
  - Access to source code.
  - The right to redistribute.
  - The right to modify.
- The license may require that modifications also be kept open.
- Most of the software in COIN-OR uses the CPL, which is a certified open-source license that is much less restrictive than the better-known GPL.
- License compatibility is an issue one has to be very careful about.

Why Open Source?

- Increases the pace of development.
- Produces more robust code.
- Introduces an inherent peer review process.
- Creates an informal reward structure.
- Creates an impetus for good documentation.
- Increases the use and distribution of code.
- Prevents obsolescence.
- Promotes reuse over reimplementation.
- Makes collaboration much easier!

"Given enough eyeballs, all bugs are shallow" –ESR

What Are the Downsides?

- Legal issues
- Initial effort is high
- Ongoing maintenance
- Funding issues
- Loss of control
- Loss of commercial opportunities
- ...

Input Data

The following are the input data needed to describe an instance of the uncapacitated facility location problem (UFL):

Data

- a set of depots $N = \{1, ..., n\}$, a set of clients $M = \{1, ..., m\}$,
- the unit transportation cost c_{ij} to service client i from depot j,
- the fixed cost f_j for using depot j

Variables

- x_{ij} is the amount of the demand for client i satisfied from depot j
- y_j is 1 if the depot is used, 0 otherwise

Mathematical Programming Formulation

The following is a mathematical programming formulation of the UFL

UFL Formulation

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j y_j && && (1)\\
\text{subject to} \quad & \sum_{j \in N} x_{ij} = d_i && \forall i \in M, && (2)\\
& \sum_{i \in M} x_{ij} \leq (\sum_{i \in M} d_i) y_j && \forall j \in N, && (3)\\
& y_j \in \{0, 1\} && \forall j \in N && (4)\\
& 0 \leq x_{ij} \leq d_i && \forall i \in M, j \in N && (5)
\end{aligned}
$$

Dynamically Generated Valid Inequalities

- The formulation presented on the last slide can be tightened by disaggregating the constraints (3).

$$x_{ij} - d_j y_j \leq 0, \forall i \in M, j \in N.$$

- Rather than adding the inequalities to the initial formulation, we can generate them dynamically.
- Given the current LP solution, x^*, y^*, we check whether

$$x^*_{ij} - d_j y^*_j > \epsilon, \forall i \in M, j \in N.$$

- We can also generate inequalities valid for generic MILPs.
- If a violation is found, we can iteratively add the constraint to the current LP relaxation.

Tightening the Initial Formulation

Here is the basic loop for tightening the initial formulation using the dynamically generated inequalities from the previous slide.

Solving the LP relaxation

1. Form the initial LP relaxation and solve it to obtain $(\hat{x}, \hat{y})$.
2. Iterate
   1. Try to generate a valid inequality violated by $(\hat{x}, \hat{y})$. If none are violated, STOP.
   2. Optionally, try to generate an improved feasible solution by rounding $\hat{y}$.
   3. Solve the current LP relaxation of the initial formulation to obtain $(\hat{x}, \hat{y})$.
   4. If $(\hat{x}, \hat{y})$ is feasible, STOP. Otherwise, go to Step 1.

Data Members

C++ Class

```
class UFL {
private:
  OsiSolverInterface * si;
  double * trans_cost; //c[i][j] -> c[xindex(i,j)]
  double * fixed_cost; //f[j]
  double * demand;     //d[j]
  int M;               //number of clients (index on i)
  int N;               //number of depots  (index in j)
  double total_demand; //sum{j in N} d[j]
  int *integer_vars;

  int xindex(int i, int j) {return i*N + j;}
  int yindex(int j)        {return M*N + j;}
};
```

Methods

C++ Class

```
class UFL {
public:
  UFL(const char* datafile);
  ~UFL();
  void create_initial_model();
  double tighten_initial_model(ostream *os = &cout);
  void solve_model(ostream *os = &cout);
};
```

Open Solver Interface

- Uniform API for a variety of solvers: CBC, CLP, CPLEX, DyLP, FortMP, GLPK, Mosek, OSL, Soplex, SYMPHONY, the Volume Algorithm, XPRESS-MP supported to varying degrees.
- Read input from MPS or CPLEX LP files or construct instances using COIN-OR data structures.
- Manipulate instances and output to MPS or LP file.
- Set solver parameters.
- Calls LP solver for LP or MIP LP relaxation.
- Manages interaction with dynamic cut and column generators.
- Calls MIP solver.
- Returns solution and status information.

Cut Generator Library

- A collection of cutting-plane generators and management utilities.
- Interacts with OSI to inspect problem instance and solution information and get violated cuts.
- Cuts include:
  - Combinatorial cuts: AllDifferent, Clique, KnapsackCover, OddHole
  - Flow cover cuts
  - Lift-and-project cuts
  - Mixed integer rounding cuts
  - General strengthening: DuplicateRows, Preprocessing, Probing, SimpleRounding

COIN LP Solver

- High-quality, efficient LP solver.
- Simplex and barrier algorithms.
- QP with barrier algorithm.
- Interface through OSI or native API.
- Tight integration with CBC (COIN-OR Branch and Cut MIP solver).

COIN Branch and Cut

- State of the art implementation of branch and cut.
- Tight integration with CLP, but can use other LP solvers through OSI.
- Uses CGL to generate cutting planes.
- Interface through OSI or native API.
- Many customization options.

The `initialize_solver()` Method

Initializing the LP solver

```
#if defined(COIN_USE_CLP)

#include "OsiClpSolverInterface.hpp"
typedef OsiClpSolverInterface OsiXxxSolverInterface;

#elif defined(COIN_USE_CPX)

#include "OsiCpxSolverInterface.hpp"
typedef OsiCpxSolverInterface OsiXxxSolverInterface;

#endif

OsiSolverInterface* UFL::initialize_solver() {
  OsiXxxSolverInterface* si =
    new OsiXxxSolverInterface();
  return si;
}
```

The create_initial_model() Method

Creating Rim Vectors

```
CoinIotaN(integer_vars, N, M * N);
CoinFillN(col_lb, n_cols, 0.0);

int i, j, index;

for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    index              = xindex(i,j);
    objective[index] = trans_cost[index];
    col_ub[index]     = demand[i];
  }
}
CoinFillN(col_ub + (M*N), N, 1.0);
CoinDisjointCopyN(fixed_cost, N, objective + (M * N));
```

The `create_initial_model()` Method

Creating the Constraint Matrix

```
CoinPackedMatrix * matrix =
   new CoinPackedMatrix(false,0,0);

matrix->setDimensions(0, n_cols);
for (i=0; i < M; i++) {  //demand constraints:
  CoinPackedVector row;
  for (j=0; j < N; j++) row.insert(xindex(i,j),1.0);
  matrix->appendRow(row);
}

for (j=0; j < N; j++) {  //linking constraints:
  CoinPackedVector row;
  row.insert(yindex(j), -1.0 * total_demand);
  for (i=0; i < M; i++) row.insert(xindex(i,j),1.0);
  matrix->appendRow(row);
}
```

Loading and Solving the LP Relaxation

Loading the Problem in the Solver

```
si->loadProblem(*matrix, col_lb, col_ub,
                objective, row_lb, row_ub);
```

Solving the Initial LP Relaxation

```
si->initialSolve();
```

The `tighten_initial_model()` Method

Tightening the Relaxation—Custom Cuts

```
const double* sol = si->getColSolution();
int newcuts = 0, i, j, xind, yind;
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    xind = xindex(i,j);  yind = yindex(j);

    if (sol[xind] - (demand[i] * sol[yind]) >
        tolerance) { // violated constraint
      CoinPackedVector cut;
      cut.insert(xind,  1.0);
      cut.insert(yind, -1.0 * demand[i]);
      si->addRow(cut,  -1.0 * si->getInfinity(), 0.0);
      newcuts++;
    }
  }
}
```

The tighten_initial_model() Method

Tightening the Relaxation—CGL Cuts

```
OsiCuts cutlist;
si->setInteger(integer_vars, N);
CglGomory * gomory = new CglGomory;
gomory->setLimit(100);
gomory->generateCuts(*si, cutlist);
CglKnapsackCover * knapsack = new CglKnapsackCover;
knapsack->generateCuts(*si, cutlist);
CglSimpleRounding * rounding = new CglSimpleRounding;
rounding->generateCuts(*si, cutlist);
CglOddHole * oddhole = new CglOddHole;
oddhole->generateCuts(*si, cutlist);
CglProbing * probe = new CglProbing;
probe->generateCuts(*si, cutlist);
si->applyCuts(cutlist);
```

The `solve_model()` Method

Calling the Solver (Built-In MIP)

```
si->setInteger(integer_vars, N);

si->branchAndBound();
if (si->isProvenOptimal()) {
  const double * solution = si->getColSolution();
  const double * objCoeff = si->getObjCoefficients();
  print_solution(solution, objCoeff, os);
}
else
  cerr << "B&B failed to find optimal" << endl;
return;
```

The solve_model() Method

Calling the Solver (CLP Requires Separate MIP)

```
CbcModel model(*si);
model.branchAndBound();
if (model.isProvenOptimal()) {
  const double * solution = model.getColSolution();
  const double * objCoeff = model.getObjCoefficients();
  print_solution(solution, objCoeff, os);
}
else
  cerr << "B&B failed to find optimal" << endl;
return;
```

Quick Introduction to CHiPPS

- CHiPPS stands for COIN-OR High Performance Parallel Search.
- CHiPPS is a set of C++ class libraries for implementing parallel or serial tree search algorithms.

What Differentiates CHiPPS?

- Intuitive interface and open source implementation.
- Very general, base classes make minimal algorithmic assumptions.
- Easy to specialize for particular problem classes.
- Designed for *parallel scalability*.
- Explicitly supports *data-intensive* algorithms.
- Operates effectively in both *parallel* and *sequential* environments.
- To our knowledge, the only other framework for general parallel tree search algorithms is the Parallel Implicit Graph Search Library (PIGSeL) by Peter Sanders.

CHiPPS Library Hierarchy

ALPS (Abstract Library for Parallel Search)

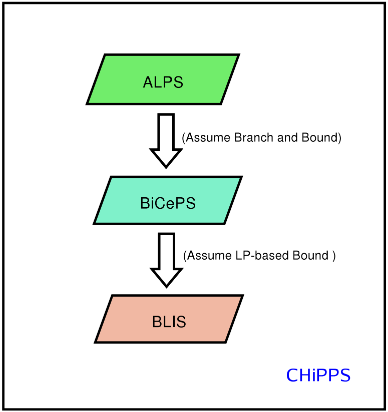
- is the search-handling layer (parallel and sequential).
- provides various search strategies.

BiCePS (Branch, Constrain, and Price Software)

- is the data-handling layer for relaxation-based optimization.
- adds notion of variables and constraints.
- assumes iterative bounding process.

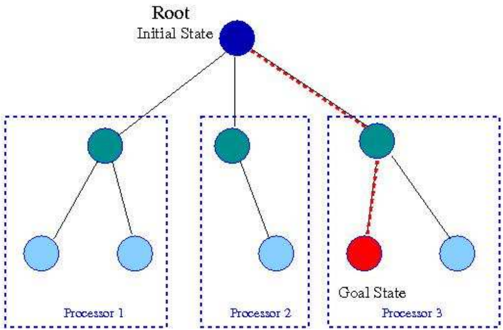
BLIS (BiCePS Linear Integer Solver)

- is a concretization of BiCePS.
- specific to models with linear constraints and objective function.



ALPS: Parallelizing Tree Search Algorithms

- The state space for a tree search can be extremely large.
- Ostensibly, tree search seems very easy to parallelize.



- *However*, the search graph may not be known a priori and there will be significant parallel overhead with naive parallelization.

ALPS: Ideas for Improving Scalability

- The design is based on a very general concept of *knowledge*.
- Knowledge is shared asynchronously through *pools* and *brokers*.
- Management overhead is reduced with the *master-hub-worker* paradigm.
- Overhead is decreased using dynamic task granularity.
- Two static load balancing techniques are used.
- Three dynamic load balancing techniques are employed.
- Use asynchronous messaging mode
- A scheduler on each process manages tasks like
  - node processing,
  - load balaning,
  - update search states, and
  - termination checking, etc.

BiCePS: Handling Data Intensive Applications

- A differencing scheme is used to store the difference between the descriptions of a child node and its parent.
- Need spend time *recovering* the explicit description of tree nodes.
- Have an option to store a explicit description when a node is at certain depth.

```
struct BcpsObjectListMod
{
    int  numRemove;
    int* posRemove;
    int  numAdd;
    BcpsObject **objects;
    BcpsFieldListMod<double> lbHard;
    BcpsFieldListMod<double> ubHard;
    BcpsFieldListMod<double> lbSoft;
    BcpsFieldListMod<double> ubSoft;
};
```

```
template<class T>
struct BcpsFieldListMod
{
    bool relative;
    int  numModify;
    int  *posModify;
    T    *entries;
};
```

BLIS: A Generic Solver for MILP

MILP

$$
\begin{aligned}
min \quad & c^T x && (6)\\
s.t. \quad & Ax \leq b && (7)\\
& x_i \in \mathbb{Z} \quad \forall\, i \in I && (8)
\end{aligned}
$$

where $(A, b) \in \mathbb{R}^{m \times (n+1)}$, $c \in \mathbb{R}^n$.

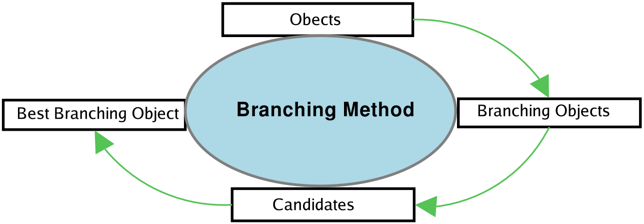
Basic Algorithmic Components

- Bounding method.
- Branching scheme.
- Object generators.
- Heuristics.

BLIS: Branching Scheme

BLIS Branching scheme comprise three components:

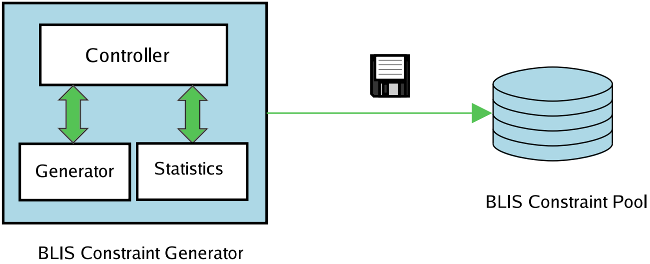
- Object: has feasible region and can be branched on.
- Branching Object:
  - is created from objects that do not lie in they feasible regions or objects that will be beneficial to the search if branching on them.
  - contains instructions for how to conduct branching.
- Branching method:
  - specifies how to create a set of candidate branching objects.
  - has the method to compare objects and choose the best one.



BLIS: Constraint Generators

BLIS constraint generator:

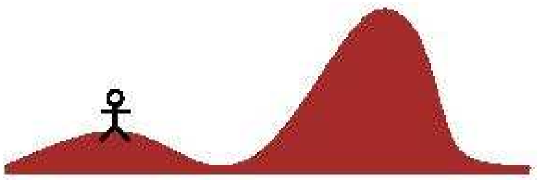
- provides an interface between BLIS and the algorithms in COIN/Cgl.
- provides a base class for deriving specific generators.
- has the ability to specify rules to control generator:
  - where to call: root, leaf?
  - how many to generate?
  - when to activate or disable?
- contains the statistics to guide generating.



BLIS: Heuristics

BLIS primal heuristic:

- defines the functionality to search for solutions.
- has the ability to specify rules to control heuristics.
  - where to call: before root, after bounding, at solution?
  - how often to call?
  - when to activate or disable?
- collects statistics to guide the heuristic.
- provides a base class for deriving specific heuristics.



ALPS Applications

What kinds of Applications?

- Constraint Programming
- Artificial Intelligence
- Discrete Optimization

Sample Applications (Matt Galati, Scott DeNegre, Yan Xu, and others)

- Knapsack Problem,
- Axial Assignment Problem,
- Steiner Problem in Graphs,
- Maxtrix Decomposition,
- Portifolio Optimization, and
- Mixed-integer Stackelberg Games, etc.

ALPS Applications: The Two Steps Required

- The first step is deriving a few classes to specify the algorithm and model.
  - `AlpsModel`
  - `AlpsTreeNode`
  - `AlpsNodeDesc`
  - `AlpsSolution`
  - `AlpsParameterSet`
- The second step is writing a `main` function.

ALPS Application: A Knapack Solver

The formulation of the binary Knapsack problem is

$$\max\{\sum_{i=1}^{m} p_i x_i : \sum_{i=1}^{m} s_i x_i \leq c, x_i \in \{0,1\}, i = 1, 2, \ldots, m\}, \tag{9}$$

First, deriving following subclasses

- `KnapModel` (from `AlpsModel`) : It stores the data used to describe a knapsack problem.
- `KnapTreeNode` (from `AlpsTreeNode`) : It defines the functions to compute path costs, expand nodes, and create children.
- `KnapNodeDesc` (from `AlpsNodeDesc`) : It stores information about which items have been fixed by branching and which are still free to select.
- `KnapSolution` (from `AlpsSolution`) It tells whether put an item in the knapsack (1) or leave it out of the knapsack (0).

ALPS Application: A Knapack Solver

Then, write a main function

```
int main(int argc, char* argv[])
{
    KnapModel model;

#if defined(SERIAL)
    AlpsKnowledgeBrokerSerial broker(argc, argv, model);
#elif defined(PARALLEL_MPI)
    AlpsKnowledgeBrokerMPI broker(argc, argv, model);
#endif

    broker.search();
    broker.printResult();
    return 0;
}
```

ALPS Application: A Knapack Solver

- Randomly generated 26 *difficult* Knapsack instances based on the rule proposed by Martello ('90).
- Tested on the SDSC Blue Gene System (Linux, MPICH, 700MHz Dual Processor, 512 MB RAM).

Table: Scalability for solving Difficult Knapsack Instances

P	Node	Ramp-up	Idle	Ramp-down	Wallclock	Eff
64	14733745123	0.69%	4.78%	2.65%	6296.49	1.00
128	14776745744	1.37%	6.57%	5.26%	3290.56	0.95
256	14039728320	2.50%	7.14%	9.97%	1672.85	0.94
512	13533948496	7.38%	4.30%	14.83%	877.54	0.90
1024	13596979694	13.33%	3.41%	16.14%	469.78	0.84
2048	14045428590	9.59%	3.54%	22.00%	256.22	0.77

BiCePS Application

What kinds of Applications?

- Discrete Optimization
- Constraint programming

Sample Applications

- Mixed Integer Linear Programming Solver (BLIS)
- Mixed Integer Quadratic Programming Solver (not implemented)
- Mixed Integer Nonlinear Programming Solver (not implemented)
- Stochastic Programming Solver (not implemented)
- Others ...

BLIS Application

What kinds of Applications?

- Mixed Integer Linear Optimization

Sample Applications (Scott DeNegre, Ted Ralphs, Yan Xu, and others)

- Vehicle Routing Problem (VRP)
- Traveling Salesman Problem (TSP)
- Mixed Integer Bilevel Solver (MiBS)
- Others ...

BLIS Applications: VRP Formulation

$$min \sum_{e \in E} c_e x_e$$

$$\sum_{e=\{0,j\} \in E} x_e = 2k, \tag{10}$$

$$\sum_{e=\{i,j\} \in E} x_e = 2 \ \ \forall i \in N, \tag{11}$$

$$\sum_{\substack{e=\{i,j\} \in E \\ i \in S, j \notin S}} x_e \geq 2b(S) \ \ \forall S \subset N, \ |S| > 1, \tag{12}$$

$$0 \leq x_e \leq 1 \ \ \forall e = \{i,j\} \in E, \ i,j \neq 0, \tag{13}$$

$$0 \leq x_e \leq 2 \ \ \forall e = \{i,j\} \in E, \tag{14}$$

$$x_e \in \mathbb{Z} \ \ \forall e \in E. \tag{15}$$

BLIS Applications: VRP

- First, derive a few subclasses to specify the algorithm and model
  - `VrpModel` (from `BlisModel`),
  - `VrpSolution` (from `BlisSolution`),
  - `VrpCutGenerator` (from `BlisConGenerator`),
  - `VrpHeurTSP` (from `BlisHeuristic`),
  - `VrpVariable` (from `BlisVariable`), and
  - `VrpParameterSet` (from `AlpsParameterSet`).
- Then, writes a `main` function

```
int main(int argc, char* argv[])
{
    OsiClpSolverInterface lpSolver;
    VrpModel model;
    model.setSolver(&lpSolver);
#ifdef  COIN_HAS_MPI
    AlpsKnowledgeBrokerMPI broker(argc, argv, model);
#else
    AlpsKnowledgeBrokerSerial broker(argc, argv, model);
#endif
    broker.search(&model);
    broker.printBestSolution();
```

- Select 16 VRP instances from public sources (Ralphs,'03)
- Tested on a Clemson Cluster (Linux, MPICH, 1654 MHz Dual Core, 4G RAM).

P	Nodes	Ramp-up	Idle	Ramp-down	Wallclock	Eff
1	40250	–	–	–	19543.46	1.00
4	36200	7.06%	7.96%	0.39%	5402.95	0.90
8	52709	9.88%	6.15%	1.29%	4389.62	0.56
16	70865	14.16%	8.81%	3.76%	3332.52	0.37
32	96160	15.85%	10.75%	16.91%	3092.20	0.20
64	163545	18.19%	10.65%	19.02%	2767.83	0.11

In October, 2007, the VRP/TSP solver won the Open Contest of Parallel Programming at the 19th International Symposium on Computer Architecture and High Performance Computing.

The CHiPPS framework is available at

`https://projects.coin-or.org/CHiPPS`

Questions? & Thank You!