SOFTWARE ENGINEERING FOR SEARCH AND OPTIMIZATION PROBLEMS

Tutorial at ECAI-2014

August, 18th 2014

Luca Di Gaspero Tommaso Urli

SaTT, Università degli Studi di Udine, Italy

ORG, NICTA, Canberra, Australia

WHO ARE WE?

- Me: Senior Lecturer of ICT at the University of Udine, Italy
  - In Optimization since 2000 (Timetabling, Scheduling, Routing, ...)
  - Involved in academic and industrial projects
  - Designer of EasyLocal, a software framework for Local Search
  - Software technology watcher
- Tommaso Urli: Post-doc researcher at the Optimisation Research Group, NICTA, Canberra, Australia
  - Ph.D. on Hybrid Metaheuristics (with Learning and CP)
  - Currently involved in Logistics and Supply Chains Optimization
  - Passionate programmer

SEARCH AND OPTIMIZATION

- Many problems in AI are solved by performing some sort of search and optimization:
  - planning, robotics, learning, constraint satisfaction (CSPs) and constrained optimization problems (COPs)
- Search itself is a hot AI topic:
  - Tree-search (SAT, Constraint Programming)
  - (Meta-)heuristic search (Hill Climbing, Simulated Annealing, Tabu Search)
  - Evolutionary algorithms
  - Swarm intelligence
- Related to (Mathematical) Optimization (OR)
  - Common problems, ``common'' search techniques
  - Cross-fertilization among fields (successful CP-AI-OR conference series)

MOTIVATIONS

- Software engineering is about (practical) methodology
- if properly applied, can simplify development process:
  - improved reliability, maintainability, replicability and reuse
  - less effort on mechanical tasks, e.g., compiling, running experiments
  - allow to focus on the important stuff, i.e., research
- more popular in industry than in academia
  - throw-away demonstrators
  - but both can equally benefit from it

OUTLINE

1. Introduction
2. Specification
3. Design and implementation
4. Validation (broad)
5. Teamwork

INTRODUCTION

According to (Sommerville 2010) FAQs

- *What are the attributes of good software?*
  Good software should deliver the *required functionality* and *performance* to the user and should be *maintainable*, *dependable* and *usable*.
- *What is software engineering?*
  Software engineering is an *engineering discipline* that is concerned with *all aspects* of software *production*.

SOFTWARE ENGINEERING
and research

- *Why should I care about software engineering in my research?*
  You might be required to develop a piece of software to provide an empirical evidence that your research ideas works. Moreover, this empirical evidence might be the only way to assess the correctness of a solution approach or to compare it with previous approaches.
- *Isn't it too sophisticated for developing research-related software?*
  It depends. Software engineering approaches can be beneficial at least in the long run, reducing the development effort (e.g., by reuse) and fostering positive research behavior (e.g., ensuring the reproducibility of computations and the comparability of results).

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
``Toy'' applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect $\neq$ Developer $\neq$ Manager $\neq$ Tester $\neq$ Customer $\neq$s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance $>$ 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
``Toy'' applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance > 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
``Toy'' applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect $\neq$ Developer $\neq$ Manager $\neq$ Tester $\neq$ Customer $\neq$s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance $>$ 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
``Toy'' applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance > 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
"Toy" applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance > 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
"Toy" applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance > 60% of overall development effort

FROM SOFTWARE PROGRAMMING TO SOFTWARE ENGINEERING

Software Programming	Software Engineering
Single developer	Teams of developers (with multiple roles)
``Toy'' applications (usually standalone)	Complex systems (integration with other systems/tools)
Short lifespan	Indefinite lifespan
Single or few stakeholders	Numerous stakeholders
Architect = Developer = Manager = Tester = Customer = User	Architect $\neq$ Developer $\neq$ Manager $\neq$ Tester $\neq$ Customer $\neq$s User
One-of-a-kind systems	System families
Built from scratch	Reuse (to amortize costs/efforts)
Minimal maintenance	Maintenance $>$ 60% of overall development effort

Process models

PROCESS MODELS

- The set of activities required to develop a software system:
  - *Specification*
  - *Design*
  - *Implementation*
  - *Validation*
  - *Deployment*
  - *Evolution*
- Many different way to organize these activities

PLAN DRIVEN AND AGILE

- Plan-driven processes: all of the process activities are planned in advance and progress is measured against this plan
- Agile processes: incremental planning, it is easier to change the process to reflect changing requirements
- In practice a *mixture* of the two

Type of models

PROCESS MODELS:
Waterfall

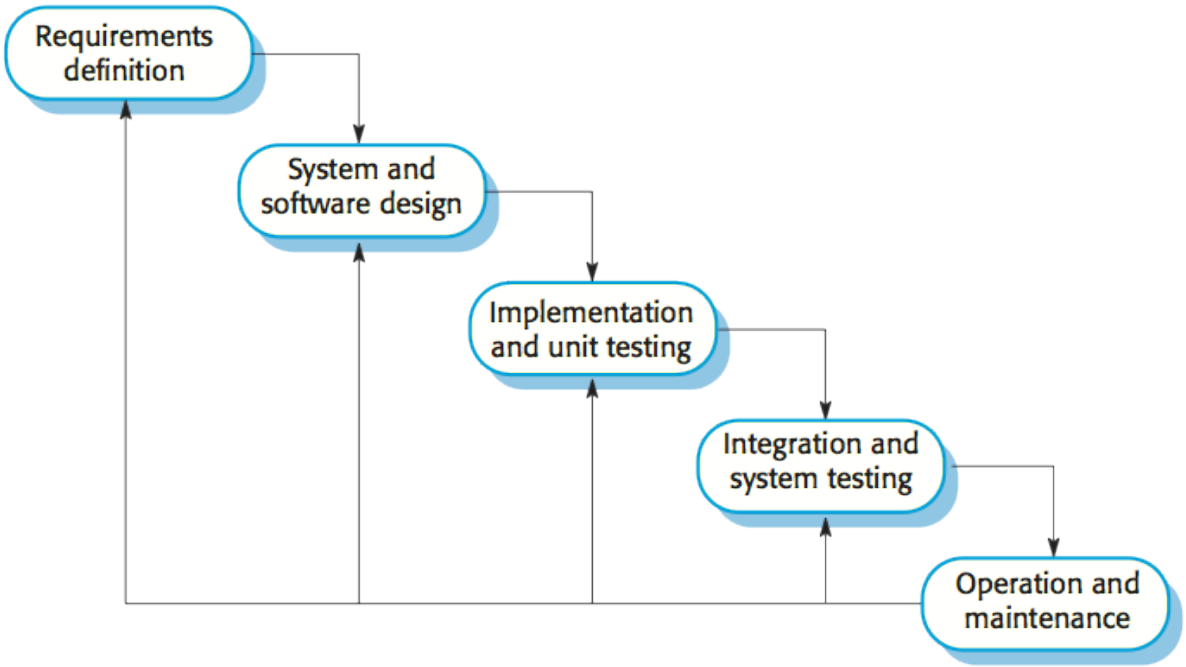


Figure: Waterfall process model (from (Sommerville 2010))

- plan-driven, distinct phases, possibly with feedback
  - classical *big-company* development

PROCESS MODELS:
Incremental development

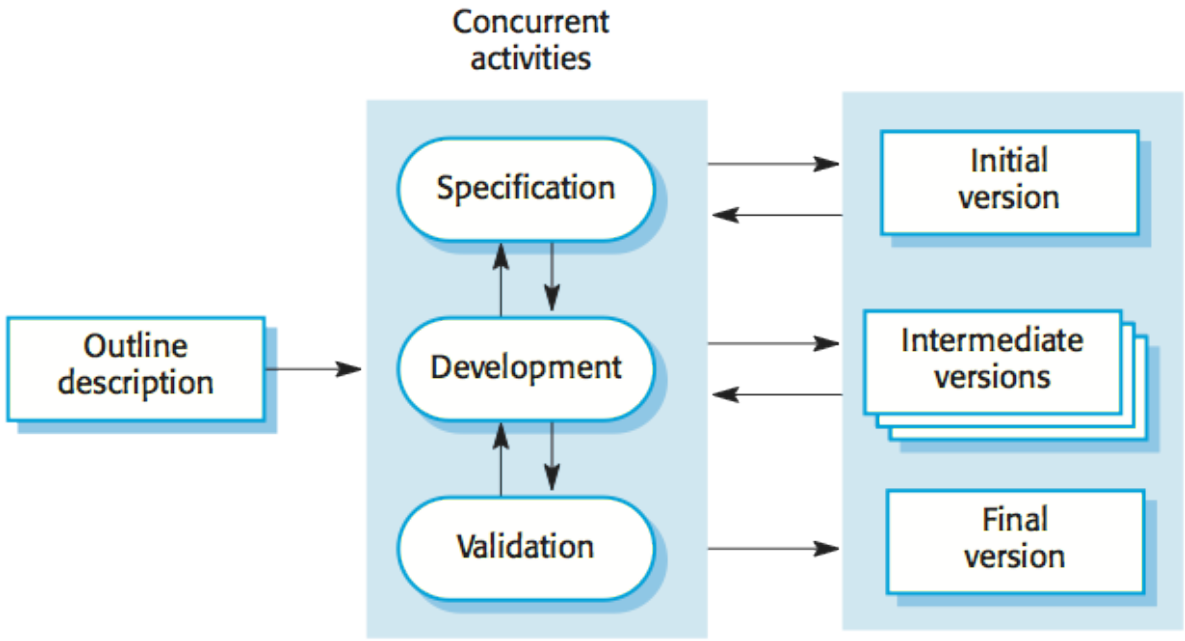


Figure: Incremental process model (from (Sommerville 2010))

- interleaved phases
- Specification is developed in conjunction with the software
  - prototypes

PROCESS MODELS:
Reuse oriented

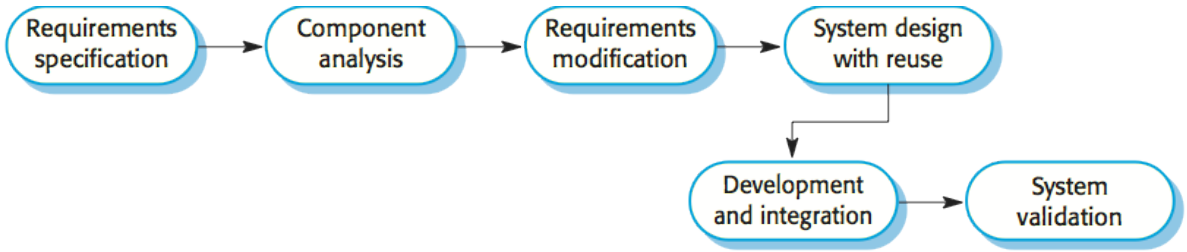
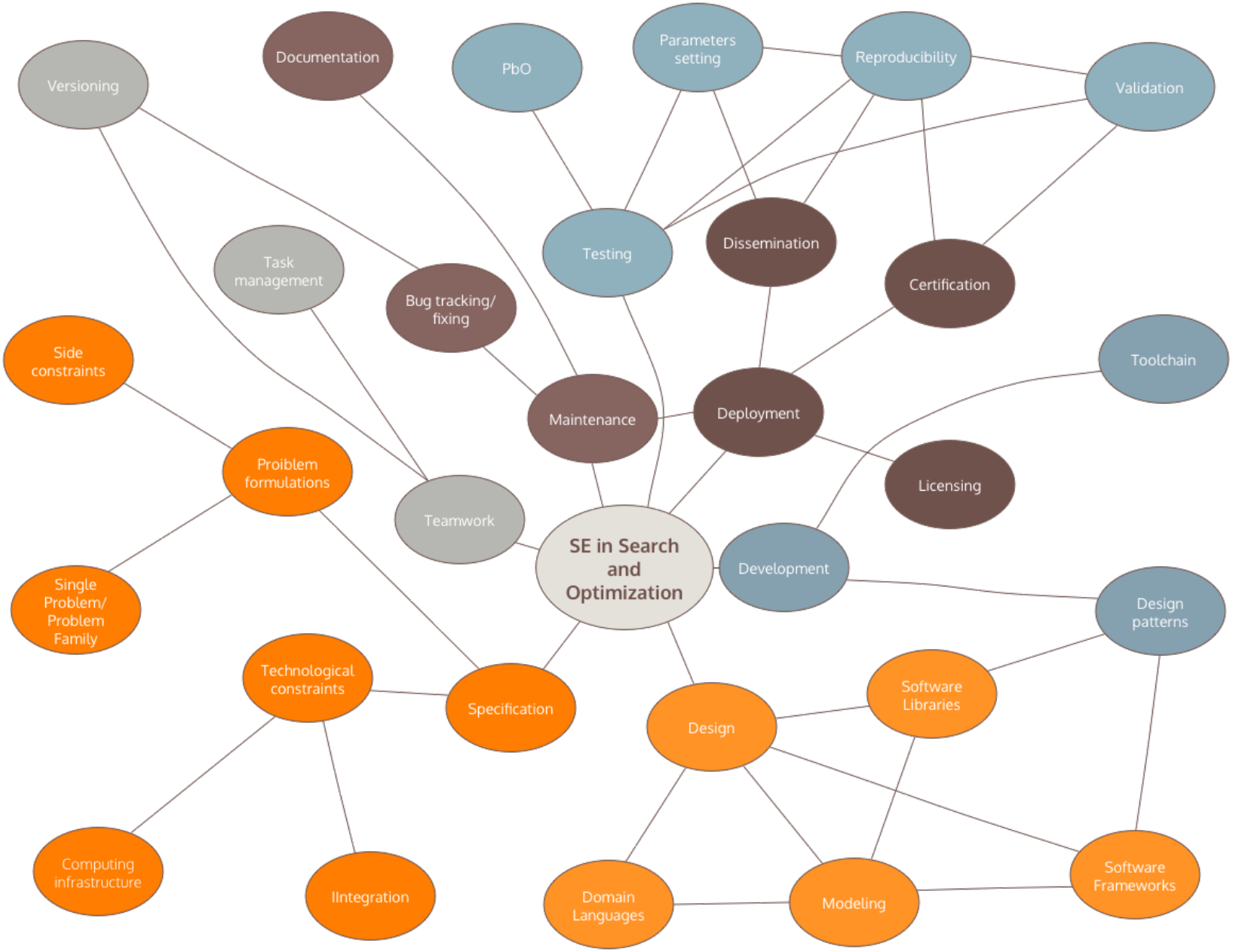


Figure: Reuse oriented process model (from (Sommerville 2010))

- assembly from existing components
  - either software components or design concepts

MINDMAP


Versioning
Documentation
PbO
Parameters setting
Reproducibility
Validation
Task management
Bug tracking/ fixing
Testing
Dissemination
Certification
Side constraints
Toolchain
Maintenance
Deployment
Proiblem formulations
Licensing
Teamwork
SE in Search and Optimization
Development
Single Problem/ Problem Family
Design patterns
Technological constraints
Specification
Design
Software Libraries
Computing infrastructure
IIntegration
Domain Languages
Modeling
Software Frameworks

SPECIFICATION

REQUIREMENT SPECIFICATION

- Might be different from standard practice in software engineering
  - e.g., depending whether or not an industrial partner is involved in the process
    - dealing with a real world problem, it might be hard(er) to elicitate the constraints and preferences in the decision maker's mind
- Differently from standard business, problems in research are usually stated in a non-ambiguous way
  - the problem you are trying to tackle can be formally stated in all its details in a problem formulation
  - nevertheless you might want to be more general and deal with multiple variants of the problem (robustness)

SINGLE PROBLEM VS PROBLEM FAMILIES

- In the real world, different variants of the same core problem usually occur
- They might differ in their essential conceptualization and/or because of side constraints
  - e.g., in Education Timetabling: (Post Enrollment, Curriculum-Based) Course Timetabling, Exam Timetabling
  - e.g., in Vehicle Routing: capacitated, time windows, heterogeneous fleet, split delivery, ...
- Solution robustness, across a family of problems, might be a requirement (or a plus if you have to publish)

REQUIREMENT SPECIFICATION

- There might be technological constraints that impose some limitations to the software
  - the computing infrastructure (e.g., parallel architectures/available libraries/operating systems)
  - the integration requirements (e.g., with the industrial partner platform or among project components)

DESIGN AND IMPLEMENTATION

Type of software tools

SOFTWARE LIBRARIES

- collections of subprograms used to develop software
- contain ``helper'' code and data, which provid services to the user's programs
- the access to the library facilities must be coded by the user, who also has to define the control logic
- *code reuse*

SOFTWARE FRAMEWORKS

- reusable designs for a software system (or subsystem)
- expressed as a set of abstract classes and their interactions for a given family of software systems
- rely on the Hollywood Principle: ``Don't call us, we'll call you'' the framework code calls the user-defined one
  - frozen spots: define the overall architecture of a software system and remain unchanged in any instantiation of the framework
  - hot spots: represent those parts where the programmers add their own code to add the functionality specific to an actual application
- *design reuse*

LANGUAGES

- programming/modeling/domain-specific languages
- ***ad hoc*** languages with a precise syntax and semantics
- contain constructs for easing either the problem modeling or the solution strategies (or both)
- need a compiler/interpreter and rely on a virtual machine that supports the language features (e.g., a constraint store for CP languages) at running time
- *knowledge reuse*

SOFTWARE TOOLS FOR SEARCH AND OPTIMIZATION
Libraries

Tool	*Reference*	*Language*	*Type*
Localizer++	(Michel and Van Hentenryck 2000)	C++	Modeling
ILOG Concert	(IBM ILOG)	C++, Java, .NET	Modeling, CP, MILP
GAlib	(Wall 1996)	C++	GA
GAUL	(Adcock 2005)	C	GA
OR-Tools	(Google Operations Research Team 2014)	C++, Python, C#	CP, LS
JSR-331	(Feldman 2012)	Java	CP
JaCoP	(Kuchcinski and Szymanek 2014)	Java	CP
Choco	(Prud'homme, Fages, and Lorca 2014)	Java	CP
Numberjack	(O'Mahony, Hebrard, and O'Sullivan 2014)	Python	CP

SOFTWARE TOOLS FOR SEARCH AND OPTIMIZATION
Frameworks

Tool	*Reference*	*Language*	*Type*
HotFrame	(Fink and Voß 2002)	C++	LS
EasyLocal++	(Di Gaspero and Schaerf 2003)	C++, Java	LS
HSF	(Dorne and Voudouris 2004)	Java	LS, GA
ParadisEO	(Cahon, Melab, and Talbi 2004)	C++	EA, LS
OpenTS	(Harder, Hill, and Moore 2004)	Java	TS
MDF	(Lau et al. 2007)	C++	LS
TMF	(Watson 2007)	C++	LS
Gecode	(Schulte, Tack, and Lagerkvist 2014)	C++	CP
HeuristicLab	(Wagner et al. 2014)	Toolkit, .NET	LS, GA, GP
Pyomo, PySP	(Hart, Watson, and Woodruff 2011)	Python	MILP, SP

SOFTWARE TOOLS FOR SEARCH AND OPTIMIZATION
Languages

Tool	*Reference*	*Language*	*Type*
SALSA	(Laburthe and Caseau 2002)	---	Language
Comet	(Van Hentenryck and Michel 2005)	---	Language
Minizinc	(Nethercote et al. 2011)	---	Language

EasyLocal

LOCAL SEARCH

- Local Search (LS) is a family of techniques for search and optimization problems.
- The techniques are non-exhaustive:
  - they **do not** guarantee to find a feasible solution;
  - they search **non-systematically** until a specific stop criterion is satisfied;
- Still attractive because
  - very flexible
  - the resulting algorithms perform fast in practice

LOCAL SEARCH BASIC COMPONENTS

- **Search Space** S: each element represent an assignment of values to the variables of $\mathcal{X}$ according to their domains $\mathcal{D}$. Should contain at least one feasible assignment.
- **Neighborhood Relation** $\mathcal{N}(s)$: how to ***move*** from a solution to a ``close'' one (usually given in an intensional fashion).
- **Cost Function** $F(s)$: assess the quality of each solution. Embeds distance from feasibility and the objective function, and drives the search towards feasible regions.

EASYLOCAL:
A C++ Object-Oriented framework for Local Search Meta-heuristics

- Fully glass-box
- Decouples control-logic from problem representation (by means of generic programming)
- Manages problem-specific features by template instantiation and class derivation
- Based on Design Patterns (Decorator, Strategy, Template and Observer) and the Hollywood principle

Motivations:

- Reuse & composition of code
- Conceptual clarity
- Prototyping and experimentation
- Design of novel techniques
- Earliest (ante litteram) adopter of some of the principles of Programming by Optimization (H. Hoos 2012)

OTHER EASYLOCAL FEATURES

- Small-sized (about 10,000 lines of code)
- Balanced use of Object-Oriented features
- Efficient implementation (5--10% loss w.r.t. direct implementation)
- The 3.0 release is ongoing (under MIT license)

Additional components/utilities:

- Command-line parser for parameters
- *Algorithm debugging* and *analysis* support:
  - Testers: basic user interface for test and debug
  - json2run: a framework for behavior analyses

Advanced LS components: - *Neighborhood combination* (union, composition) - *High-level solving strategies* (Token Ring, ILS, VNS, etc.)

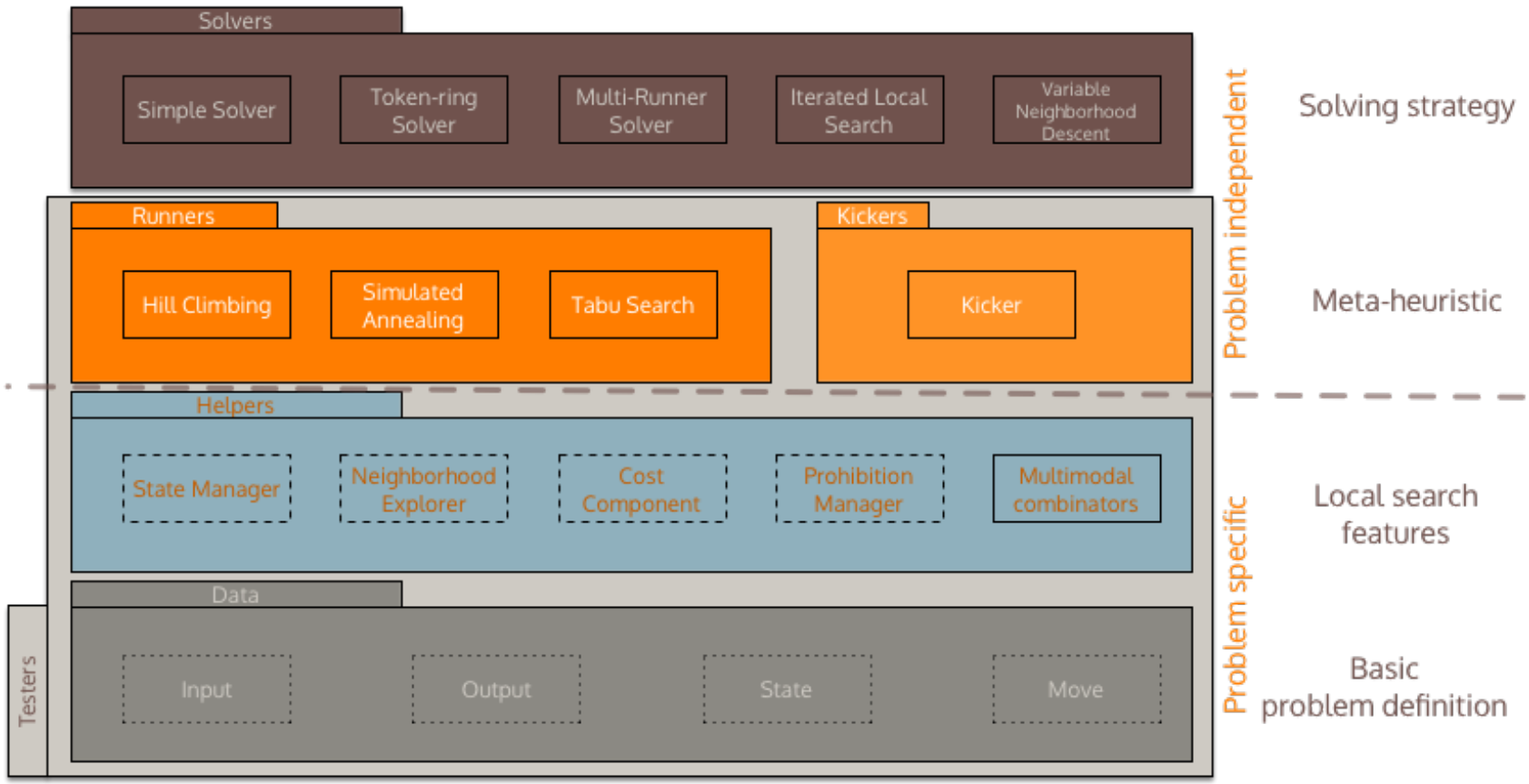
LOCAL SEARCH CONCEPTUALIZATION

- *Search Space* S: each element represents a possible solution of the problem
- *Neighborhood Relation* $N(s)$: how to move from a solution to a ``close'' one (usually given in an intensional fashion)
- *Cost Function* $F(s)$: assesses the quality of each solution, embedding also distance from feasibility

```
procedure LocalSearch(S, N, F)
begin
  s_0 := InitialSolution();
  i := 0;
  while (¬StopSearch(s_i, i)) do
    m := SelectMove(s_i, F, N);
    if (AcceptableMove(m, s_i, F)) then
      s_{i+1} := s_i ⊕ m
    end if;
    i := i + 1
  end while
end procedure
```

EASYLOCAL ARCHITECTURE

- Main *hotspots*:
  - Data classes (template instatiation)
  - Helpers (class derivation)



AN EXAMPLE OF EASYLOCAL ABSTRACT CODE

```
procedure LocalSearch(S, N, F)
begin
  s_0 := InitialSolution(); i
       := 0;
  while (¬StopSearch(s_i, i)) do
    m := SelectMove(s_i, F, N);
    if (AcceptableMove(
          m, s_i, F)) then
      s_{i+1} := s_i ⊕ m
    end if;
    i := i + 1
  end while
end procedure
```

```
template <class Input, class State, class
      Move>
void MoveRunner<Input,State,Move>::Go()
{
  InitializeRun();
  while (!StopCriterion()) {
    SelectMove();
    if (AcceptableMove())

      MakeMove();

    UpdateIterationCounter();
  }
}
```

META-INSTANTIATING THE ABSTRACT ALGORITHM

Hill Climbing

```
template <class Input, class State, class Move>
bool HillClimbing<Input,State,Move>::AcceptableMove()
{ return (current_move_cost <= 0); }
```

Simulated Annealing

```
template <class Input, class State, class Move>
bool SimulatedAnnealing<Input,State,Move>::AcceptableMove()
{ return (current_move_cost <= 0) ||
    (Random::Float(0.0, 1.0) < exp(-current_move_cost/temperature)); }
```

Tabu Search

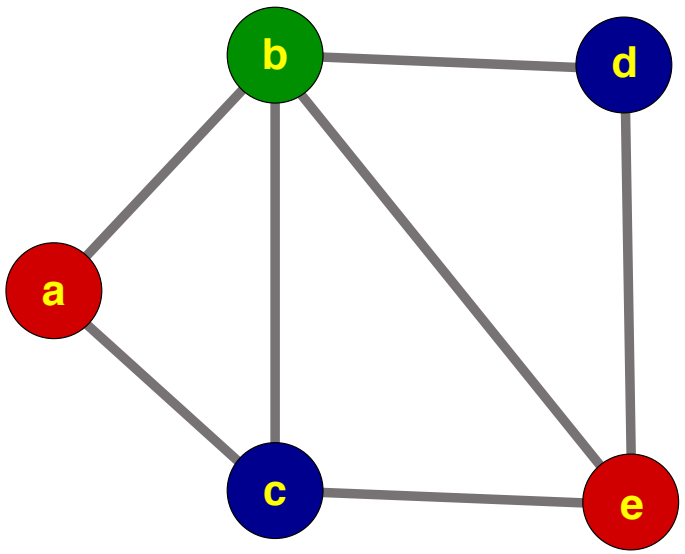
```
template <class Input, class State, class Move>
bool TabuSearch<Input,State,Move>::AcceptableMove()
{ return !tabu_list.member(current_state, current_move) || tabu_list.
     Aspiration(current_state, best_state, current_move,
     current_move_cost); }
```

EASYLOCAL DEVELOPMENT FLOW

1. Define the basic Data classes: Input, Output, State and Move classes
2. Define the helper classes and fill-in the hot spots methods:
   - The strategy for constructing a state: StateManager::RandomState()
   - The cost evaluation: CostComponent::ComputeCost()
   - The neighborhood exploration strategies: NeighborhoodExplorer:: FirstMove(), NextMove(), MakeMove()
   - The incremental cost evaluation: DeltaCostComponent::ComputeDeltaCost()
   - Other suitable helpers: TabuList::Inverse()
3. Instantiate Runners and Solvers in a main program driver

A CASE STUDY: K-GRAPHCOLORING

Given a graph $G = (V, E)$, and the set $\{0, \ldots, k-1\}$ of color values, find an assignment of color to vertices such that adjacent vertices are assigned different colors



- Variables: $c_v, v \in V$,
- Domains: $D_v = \{0, \ldots, k-1\}$,
- Constraints: $\forall (u, v) \in E \quad c_u \neq c_v$

EASYLOCAL K-GRAPHCOLORING
Input

- Input: an undirected graph $G = (V, E)$, the cardinality k of the set of colors

```
class GraphCol
{
public:
  graph G;
  unsigned int k;
};
```

EASYLOCAL K-GRAPHCOLORING
State

- State: a map $c : V \to \{0, \ldots, k-1\}$ from vertices to colors which represent the assignment of values to the c_v variables

```
class Coloring
{
   ...
public:
  const GraphCol& in;
  std::map<graph::node, unsigned int> color;
  std::set<graph::node> conflicts;
   ...
};
```

EASYLOCAL K-GRAPHCOLORING
State Manager

- StateManager::RandomState(): a random assignment of k colors to vertices

```
class ColoringManager : public StateManger<GraphCol, Coloring, int>
{
public:
   ...
  void RandomState(Coloring& c) const
  {
    for (const graph::node& v : c.G.nodes)
      c.color[v] = Random::Int(0, in.k - 1);

    for (const graph::edge& e : c.G.edges)
      if (c.color[e.from] == c.color[e.to])
      {
        conflicts.insert(e.from);
        conflicts.insert(e.to);
      }
  }
};
```

EASYLOCAL K-GRAPHCOLORING
Cost Function

- Cost Function: accounts for the ***conflicting*** edges

$$F(c) = |\{(u, v) \in E : c(u) = c(v)\}|$$

```
class EdgeConflict : public CostComponent<GraphCol, Coloring, int>
{
public:
   ...
  int ComputeCost(const Coloring& c) const
  {
    int cost = 0;
    for (const graph::edge& e : c.G.edges)
      if (c.color[e.from] == c.color[e.to])
        cost++;

    return cost;
  }
};
```

EASYLOCAL K-GRAPHCOLORING
Move

- Move: change the color of one node $m = \langle v, c_{old}, c_{new} \rangle$

```
class Recolor
{
public:
  graph::node v;
  unsigned int c_old, c_new;
};
```

EASYLOCAL K-GRAPHCOLORING
Neighborhood Explorer

- Move selection: focus on conflicting nodes only

```
class RecolorExplorer : public NeighborhoodExplorer<GraphCol, Coloring,
    Recolor, int>
{
public:
  ...
  void FirstMove(const Coloring& c, Recolor& rc) const throw (
      EmptyNeighborhood)
  {
    if (c.conflicts.empty())
      throw EmptyNeighborhood();
    rc.v = *c.conflicts.begin();
    rc.c_old = c.color[rc.v];
    rc.c_new = rc.c_old == 0 ? 1 : 0;
  }
```

EASYLOCAL K-GRAPHCOLORING
Neighborhood Explorer

- Move selection: focus on conflicting nodes only

```
bool NextMove(const Coloring& c, Recolor& rc) const
{
  rc.c_new = (rc.c_new + 1) % in.k;
  if (rc.c_new == rc.c_old)
  {
    std::set<graph::nodes>::const_iterator it;
    for (it = c.conflicts.begin(); it != c.conflicts.end(); it++)
      if (*it == rc.v)
        break;
    it++;
    if (it == c.conflicts.end())
      return false;
    rc.v = *it;
    rc.c_old = c.color[rc.v];
    rc.c_new = (rc.c_old + 1) % in.k;
  }
  return true;
}
```

EASYLOCAL K-GRAPHCOLORING
Neighborhood Explorer

- **Move selection:** focus on conflicting nodes only

```
void MakeMove(Coloring& c, const Recolor& rc) const
{
  c.color[rc.v] = rc.c_new;
  std::set<graph::nodes> to_check;
  for (const graph::node& v : rc.n.adjacent)
  {
    if (c.color[v] == rc.c_new)
      c.conflicts.insert(v);
    if (c.color[v] == rc.c_old)
    {
      c.conflicts.erase(v);
      to_check.insert(v);
    }
  }
  for (const graph::node& v : to_check)
    for (const graph::node& w : v.adjacent)
      if (w != rc.v && c.color[w] == c.color[v])
      {
        c.conflict.insert(v);
        break;
      }
}
```

Gecode

GECODE

Gecode is an open, free, portable, accessible, and efficient environment for developing constraint-based systems and applications (Schulte, Tack, and Lagerkvist 2014).

open	interfacing to other systems, programming of new propagators (i.e., implementation of constraints), branching strategies, and search engines, introducing new variables
comprehensive	constraints over integers, booleans, sets, and floats, more than 70 global constraints, many branching heuristics, search engines, ...
free	MIT license
portable	written in standard C++
accessible	well documented and with many examples
efficient	won MiniZinc Challenge in 2008--2012, as fast as commercial products
parallel	builtin support for multithreading
alive	frequently updated, new releases every 2-3 months (now version 4.2)

GECODE DEVELOPMENT FLOW

1. Define a Input class
2. Derive a Model class from the suitable Space subclass and fill-in the hot spots methods:
   - The model posting: in the class constructor (optionally by passing an option to it)
   - The clone constructor: Model(bool, const Model&)
   - The copy method: Space* copy(bool)
   - The solution printing: void print(ostream&)
3. Instantiate the model in a program driver and/or through scripting

K-GRAPHCOLORING IN GECODE

```
class GraphColor : public Script
{
protected:
  /// Graph
  std::shared_ptr<GraphCol> gc;
  /// Color of nodes
  IntVarArray color;
public:
  /// Branching to use for model
  enum {
    BRANCH_DEGREE, ///< Choose variable with largest degree
    BRANCH_SIZE ///< Choose variable with smallest size
  };
  /// The actual model
  GraphColor(const InstanceOptions& opt)
  {
   std::vector<std::string> res;
   StringSplit(opt.instance(), ":", res);
    gc = std::make_shared<GraphCol>(res[0], atoi(res[1].c_str()));
    color = IntVarArray(*this, gc->n_nodes, 0, gc->k);
    for (int u = 0; u < gc->n_nodes; u++)
      for (int v : gc->edges[u])
        rel(*this, color[u] != color[v]);
```

K-GRAPHCOLORING IN GECODE

```
  switch (opt.branching())
  {
    case BRANCH_DEGREE:
      branch(*this, color, tiebreak(INT_VAR_DEGREE_MAX(),
           INT_VAR_SIZE_MIN()), INT_VAL_MIN());
      break;
    case BRANCH_SIZE:
      branch(*this, color, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
      break;
  }
}
```

K-GRAPHCOLORING IN GECODE

```
  /// Constructor for cloning \a s
  GraphColor(bool share, const GraphColor& s) : Script(share, s), gc(s.
      gc)
  {
    color.update(*this, share, s.color);
  }
  /// Copying during cloning
  virtual Space* copy(bool share)
  {
    return new GraphColor(share, *this);
  }
  /// Print the solution
  virtual void print(std::ostream& os) const
  {
    os << color << std::endl;
  }
};
```

K-GRAPHCOLORING IN GECODE

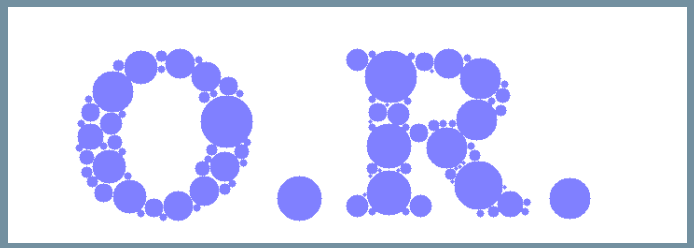
```
int main(int argc, char* argv[])
{
  InstanceOptions opt("GraphColor");

  opt.icl(ICL_DOM);
  opt.solutions(1);
  opt.branching(GraphColor::BRANCH_DEGREE);
  opt.branching(GraphColor::BRANCH_DEGREE, "degree");
  opt.branching(GraphColor::BRANCH_SIZE, "size");
  opt.parse(argc, argv);

  Script::run<GraphColor, DFS, InstanceOptions>(opt);
  return 0;
}
```

OR-tools

GOOGLE OR-TOOLS



Or-Tools is a suite of software tools for Constraint Programming and Operations Research problems with interfaces in C++, Python, Java, and .NET

- a constraint programming solver
- a wrapper around multiple (integer) linear programming solvers
- a set of libraries for
  - vehicle routing, knapsack, graphs

OR-TOOLS DEVELOPMENT FLOW
CP

1. Define a Input class
2. Create a Solver object
3. Define the variables
4. Post the constraints
5. Define a Decision builder
6. Prepare the solver for the search
7. Run the search

K-GRAPHCOLORING IN OR-TOOLS

```
from google.apputils import app
import gflags
import random
from ortools.constraint_solver import pywrapcp

FLAGS = gflags.FLAGS
gflags.DEFINE_string('instance', None, 'path␣to␣instance␣file')
gflags.DEFINE_integer('k', None, '(max)␣number␣of␣colors')
gflags.DEFINE_string('mode', 'CP', 'mode␣(LS␣or␣CP)')

class GraphCol:
    def __init__(self, filename, k):
        self.k = k
...
```

K-GRAPHCOLORING IN OR-TOOLS
The CP part

```
def main(unused_argv):
  gc = GraphCol(FLAGS.instance, FLAGS.k)
  solver = pywrapcp.Solver('k-GraphColoring')
  color = [solver.IntVar(0, gc.k - 1, 'color_' + str(u)) for u in range(
       gc.n_nodes)]
  for u, adj in gc.edges.iteritems():
      for v in adj:
          solver.Add(color[u] != color[v])
  db = solver.Phase(color, solver.CHOOSE_MIN_SIZE_LOWEST_MIN, solver.
       ASSIGN_MIN_VALUE)
  solver.NewSearch(db)
  solver.NextSolution()
```

OR-TOOLS DEVELOPMENT FLOW
Local Search

1. Define a Input class
2. Create a Solver object
3. Define the variables
4. Define a Local Search operator class
   - the method OnStart() initializes the neighborhood exploration
   - the method OneNeighbor() performs the move and prepares the next neighborhood
5. Prepare an initial solution
6. Prepare the solver for the search
7. Run the search

K-GRAPHCOLORING IN OR-TOOLS
The Local Search part

```
class Recolor(pywrapcp.IntVarLocalSearchOperator):
    """Change␣the␣color␣of␣one␣node."""

    def __init__(self, gc, variables):
        pywrapcp.IntVarLocalSearchOperator.__init__(self, variables)
        self.__gc = gc

    def OnStart(self):
        self.__conflicts = {}
        for u in range(self.__gc.n_nodes):
            for v in self.__gc.edges[u]:
                if self.Value(u) == self.Value(v):
                    self.__conflicts[u] = True
                    self.__conflicts[v] = True
        self.__index = 0
        self.__v = self.__conflicts.keys()[self.__index]
        self.__c_old = self.Value(self.__v)
        self.__c_new = 0 if self.__c_old != 0 else 1
```

K-GRAPHCOLORING IN OR-TOOLS

The Local Search part

```
def OneNeighbor(self):
    if self.__index == len(self.__conflicts):
        return False
    self.SetValue(self.__v, self.__c_new)
    self.__c_new = (self.__c_new + 1) % self.__gc.k
    if self.__c_new == self.__c_old:
        self.__index = self.__index + 1
        if self.__index < len(self.__conflicts):
            self.__v = self.__conflicts.keys()[self.__index]
            self.__c_old = self.Value(self.__v)
            self.__c_new = (self.__c_old + 1) % self.__gc.k
    return True

def IsIncremental(self):
    return False
```

K-GRAPHCOLORING IN OR-TOOLS

The Local Search part

```
conflicts = [(color[u] == color[v]).Var() for u, adj in gc.edges.
    iteritems() for v in adj]
sum_conflicts = solver.Sum(conflicts)
obj = solver.Minimize(sum_conflicts, 1)
recolor = Recolor(gc, color)
first_solution = solver.Assignment()
first_solution.Add(color)
for u in range(gc.n_nodes):
   first_solution.SetValue(color[u], random.randint(0, gc.k - 1))

ls_params = solver.LocalSearchPhaseParameters(recolor, None)
ls = solver.LocalSearchPhase(first_solution, ls_params)
collector = solver.LastSolutionCollector()
collector.Add(color)
collector.AddObjective(sum_conflicts)
solver.Solve(ls, [collector, obj])
```

Minizinc

MINIZINC

MiniZinc is a medium-level constraint modelling language. It is high-level enough to express most constraint problems easily, but low-level enough that it can be mapped onto existing solvers easily and consistently.

MINIZINC DEVELOPMENT FLOW

1. Define a model
2. Define a data file to instantiate the input variables
3. Run the search

MINIZINC GRAPHCOLORING
Problem model

```
int: n;
int: c;

array [1..n,1..n] of int: E;

array [1..n] of var 1..c: Col;

constraint
  forall (i in 1..n, j in i+1..n)
     (if E[i,j] = 1 then Col[i] != Col[j] else true endif);

solve satisfy;

output [show(Col)]
```

MINIZINC GRAPHCOLORING
Data file

```
n = 8;
c = 3;

% 1 Friuli Venezia Giulia
% 2 Veneto
% 3 Trentino Alto Adige
% 4 Lombardy
% 5 Emilia-Romagna
% 6 Piedmont
% 7 Liguria
% 8 Aosta Valley

E = [|0,1,0,0,0,0,0,0
    |1,0,1,1,1,0,0,0
    |0,1,0,1,0,0,0,0
    |0,1,1,0,1,1,0,0
    |0,1,0,1,0,1,1,0
    |0,0,0,1,1,0,1,1
    |0,0,0,0,1,1,0,0
    |0,0,0,0,0,1,0,0|];
```

Prototyping and incremental delivery

(FAST) PROTOTYPING

- Rapid software development to validate requirements
- Help users and developers to understand requirements
  - users can experiment with the system in order to see how it works
  - the prototype can help in eliciting real world constraints/preferences that are in the decision maker's mind
- Some functionalities might be left out
  - it should focus on areas that are not well-understood
  - error checking and recovery might be ignored
  - focus on functional requirements

INCREMENTAL DELIVERY

- The development is split into increments, each of them delivering parts of the functionalities (e.g., capturing a subset of constraints or business rules)
  - the standard approach in agile methods
- Requirements are assigned a priority, and the earliest development focuses on highest priorities
- System functionality can be validated sooner
- Each increment is in fact a prototype and helps eliciting requirements for later increments

PROCESS MODEL:
Incremental Delivery

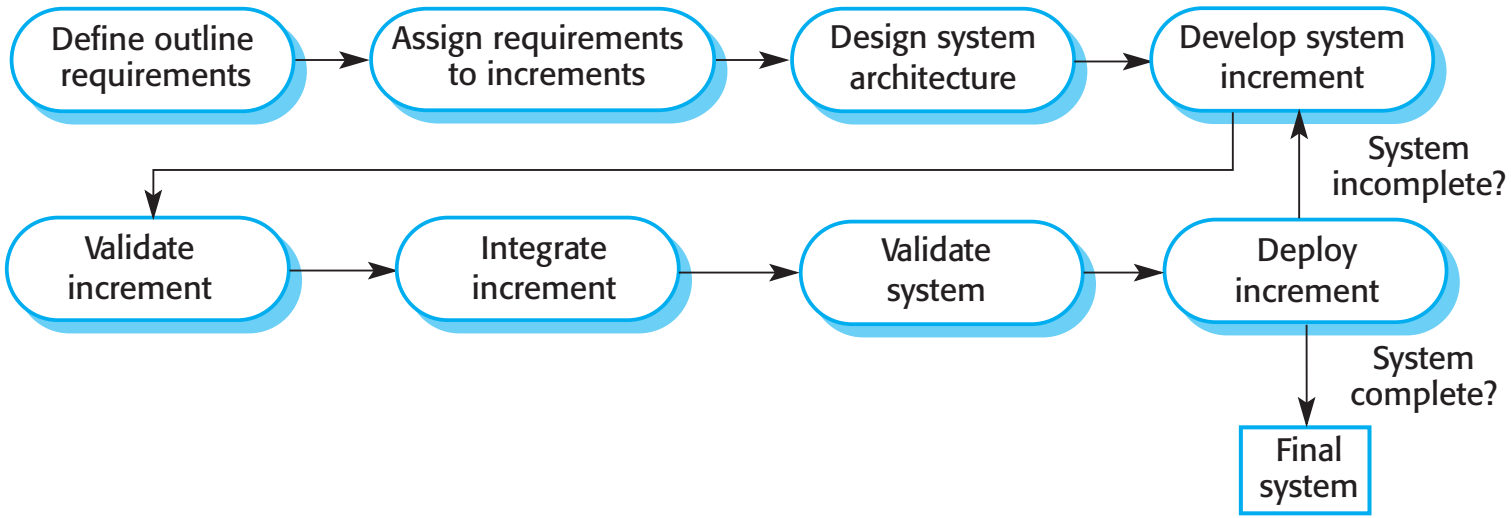
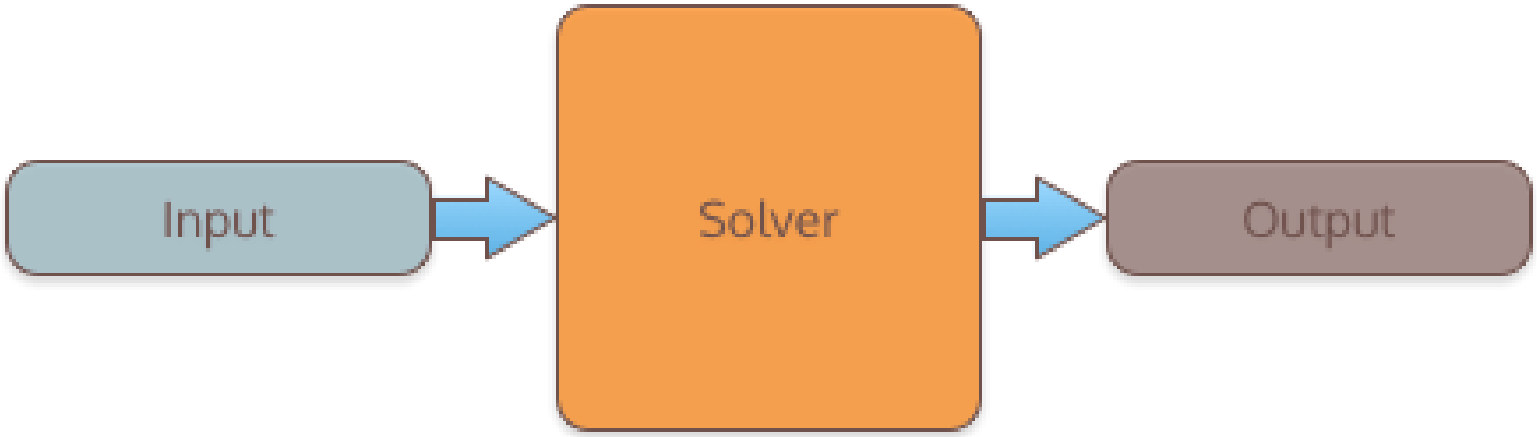


Figure: Incremental delivery (from (Sommerville 2010))

Input-Output formats

INPUT-OUTPUT FORMATS



- a relevant issue in many aspects: integration, validation, comparison
  - text-based delimited file
  - XML
  - JSON
  - database

INPUT-OUTPUT FORMATS
Text-based delimited file

CVRP

```
NAME : A-n39-k5
COMMENT : (Augerat et al, Min no
     of trucks: 5, Optimal value:
     822)
TYPE : CVRP
DIMENSION : 39
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 100
NODE_COORD_SECTION
 1 9 35
 2 43 19
 ...
DEPOT_SECTION
 1
 -1
EOF
```

CVRPTW

```
 0 50 75 0 9999 9999 9999 0 0 0
 1 49 81 0 9999 9999 9999 10 0 0
 2 51 81 0 9999 9999 9999 10 0 0
 3 47 83 0 9999 9999 9999 10 0 0
 4 53 83 0 9999 9999 9999 10 0 0
 5 49 85 0 9999 9999 9999 10 0 0
 ...
```

Pros: simple to parse and handle

Cons: not self-explaining, not too flexible, different formats for the same problem (or variants)

INPUT-OUTPUT FORMATS
XML

```
<?xml version="1.0" encoding=
    "utf-8"?>
<ConcretePlanning>
  <Orders>
    <Order code="4">
      <ConstructionYard code=
          "4">
        <WaitingMinutes>5</
            WaitingMinutes>
        ...
```

```
...
  <Vehicle code="TEST" type="Pump"
      availableFrom="2011-01-10
      T14:45:26+01:00">
    <PumpLineLength>10</
        PumpLineLength>
    <NormalVolume>11</NormalVolume>
    <MaximumVolume>13</MaximumVolume>
    <DischargeM3PerHour>8</
        DischargeM3PerHour>
  </Vehicle>
...
</ConcretePlanning>
```

Pros: human-readable/self-explaining, flexible and extensible, handles problem variants

Cons: needs specific librararies for parsing/generating, verbose

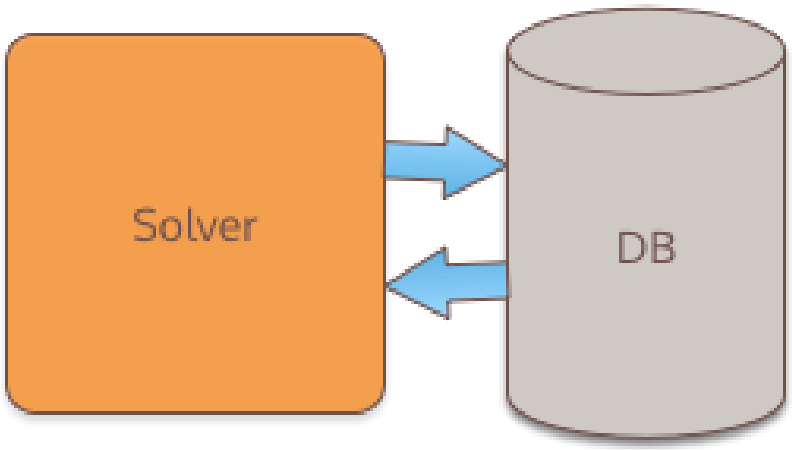
INPUT-OUTPUT FORMATS
JSON


```
{
    "activities": [
        {
            "act_0_9": {
                "duration": 60,
                "time_window": {
                    "start": "2014-03-04T08:00:00",
                    "finish": "2014-03-04T18:00:00"
                },
                "required_operators": 1,
                "location": {
                    "latitude": 45.96156006381324,
                    "longitude": 13.387064743048473
                },
                "patient_name": "Tonya England"
            },
    ...
```


Pros: lightweight, self-explaining, flexible and extensible, handles problem variants

Cons: not widespread

INPUT-OUTPUT FORMATS
Database



Pros: direct access to problem data (no intermediate format)

Cons: heavyweight, depending on the data model (i.e., relational) might be quite unflexible

VALIDATION (BROAD)

VALIDATION

- Validation, in essence, is the activity that aims at verifying whether the requirements are met
- in a incremental delivery perspective can start very soon
- in optimization research means also running your algorithms/solvers
  - for eliciting ``hidden'' constraints in real world problems
  - for empirical analysis

EXPERIMENTAL ALGORITHMICS

D. Johnson's ``A Theoretician's Guide to the Experimental Analysis of Algorithms''

- Do publishable work:
  - Tie your paper to the literature
    (if your work is new, create benchmarks)
  - Use instance testbeds that support general conclusions
  - Ensure comparability
- Convincing
  - Statistics and data analysis techniques
  - Ensure reproducibility
  - Report the full story
  - Draw well-justified conclusions and look for explanations
  - Present your data in informative ways

EXPERIMENTAL ALGORITHMICS

D. Johnson's ``A Theoretician's Guide to the Experimental Analysis of Algorithms''

- Do publishable work:
  - Tie your paper to the literature
    (if your work is new, create benchmarks)
  - Use instance testbeds that support general conclusions
  - Ensure comparability
- Convincing
  - Statistics and data analysis techniques
  - Ensure reproducibility
  - Report the full story
  - Draw well-justified conclusions and look for explanations
  - Present your data in informative ways

PROBLEM VALIDATORS

- (Third-party) Certification of your (optimization) results
  - Input validators: take an input instance and check whether it is consistent (or manifestly infeasible)
  - Output validators: take an input instance and an output solution and check whether the solution is feasible (and its cost)

The Problem Management System

A PAPER EXAMPLE

The Stand-Alone paper: *Solving the X Problem Using Technique Y*

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Y
- Conclusions

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Y
- Conclusions

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Z
- Conclusions:

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Z
- Conclusions:

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Z
- Conclusions:

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Z
- Conclusions:

A PAPER EXAMPLE

The Stand-Alone paper: ***Solving the X Problem Using Technique Y***

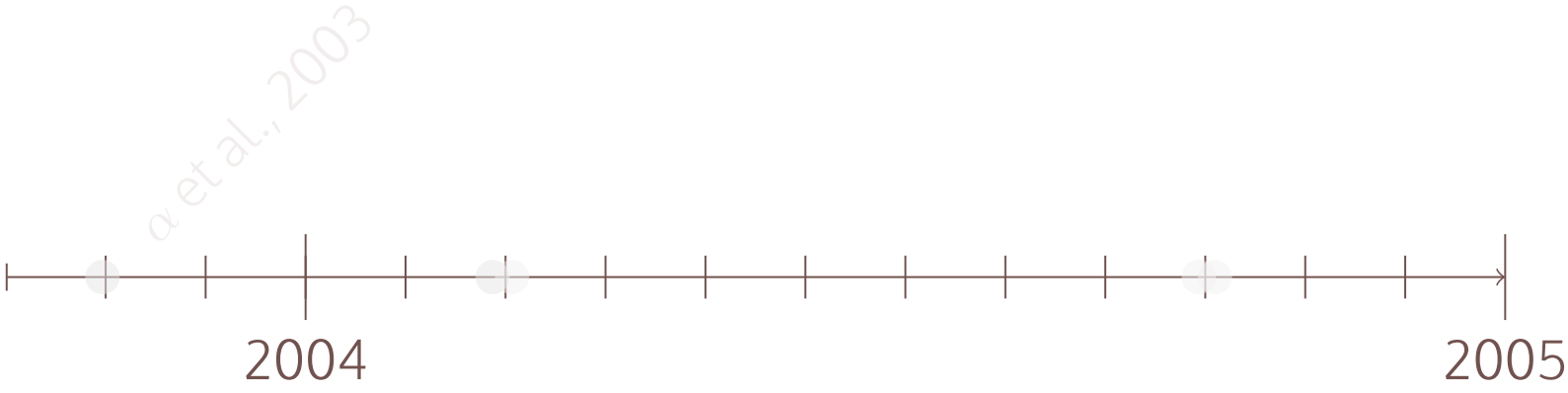
- Intro: a plotless sequence of shallow references
- Modeling X: a semi-formal formulation
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with their implementation of technique Z
- Conclusions: we did a good job!

A PAPER EXAMPLE

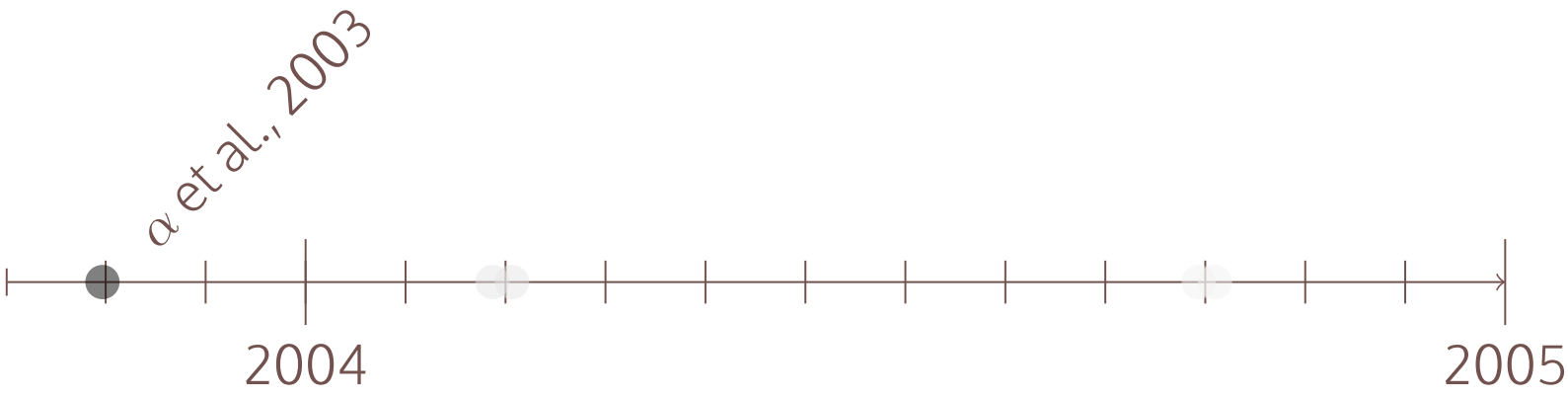
The Stand-Not-So-Alone paper: ***Solving the X Problem Using Technique Y***

- Intro: a plotless sequence of shallow references
- Modeling X: as a variant of X'
- Adaptation of Y to solve X
- Description of (a few) instances of X
- Experimental analysis: comparison
  - with the manual solution of X
  - between Y_1, Y_2, and Y_3 (parameter tuning)
  - with the literature on X' (adapting Y to X')
- Conclusions: ?

A CORRESPONDENCE EXAMPLE

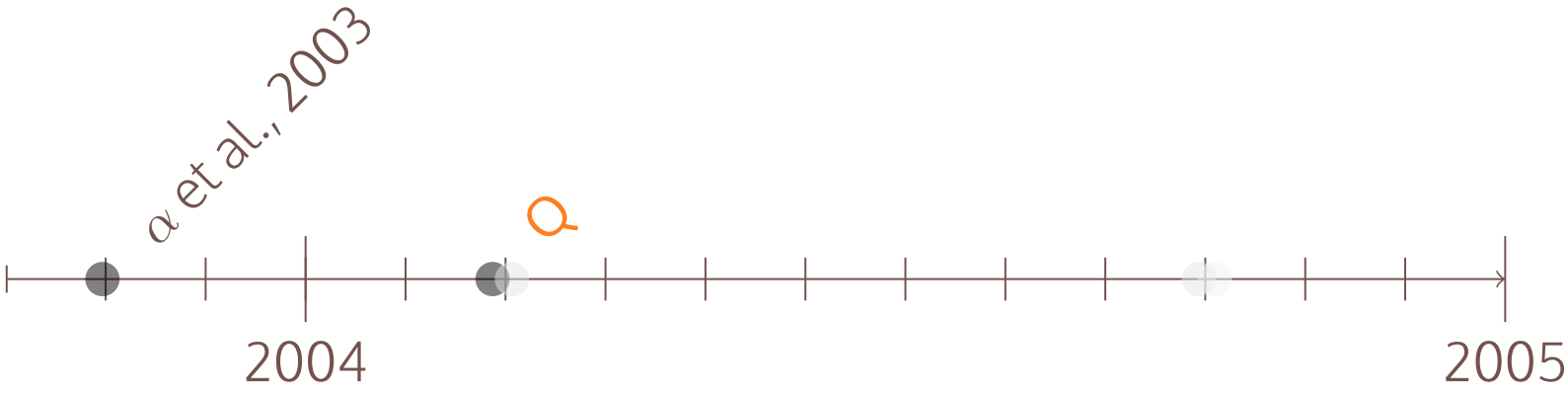


A CORRESPONDENCE EXAMPLE



Prologue: α et al. (2003) published a paper on problem X with outstanding results.

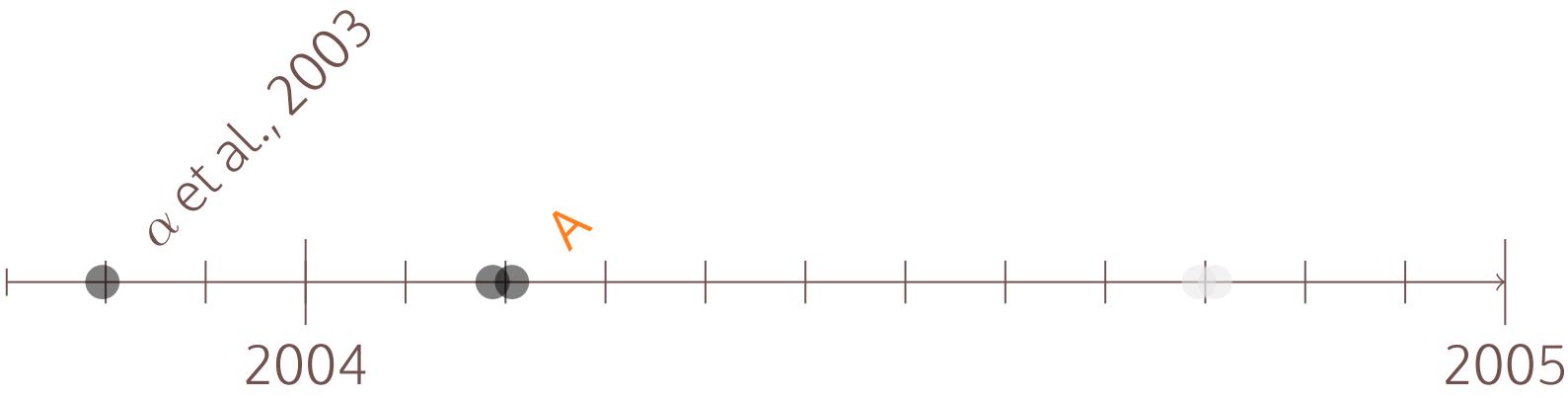
A CORRESPONDENCE EXAMPLE



Q: Dear α,
I am writing you to ask if you can send me a copy of the solution data of the outstanding results you published in your recent paper on problem X. All the best,
β

A: Hi β,
I left the University a few months ago. I do not have the data handy but I will look for them in a backup.
Kind regards, α

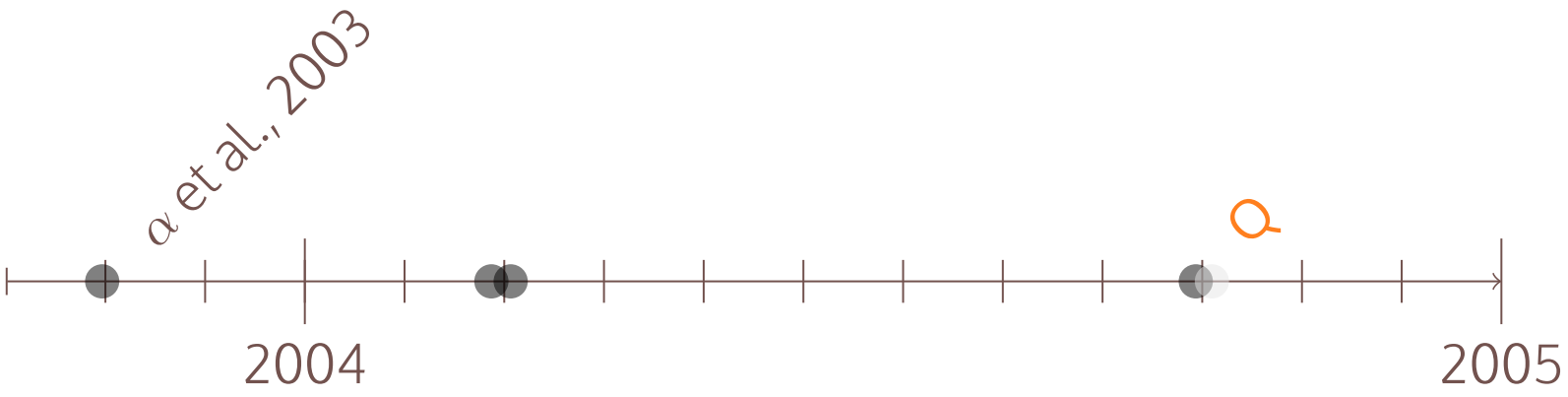
A CORRESPONDENCE EXAMPLE



Q: Dear α,
I am writing you to ask if you can send me a copy of the solution data of the outstanding results you published in your recent paper on problem X. All the best,
β

A: Hi β,
I left the University a few months ago. I do not have the data handy but I will look for them in a backup.
Kind regards, α

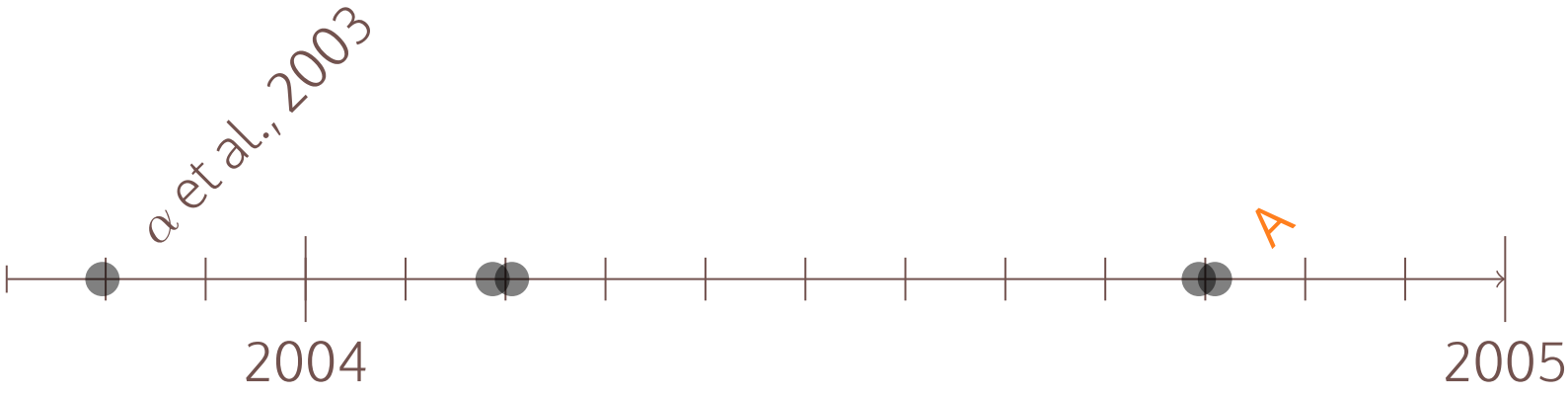
A CORRESPONDENCE EXAMPLE



Q: Dear α,
Could you find the data? β

A: Hi β,
Unfortunately the backup was corrupted and at present I cannot run again the experiments to obtain that data. α

A CORRESPONDENCE EXAMPLE



Q: Dear α,
Could you find the data? β

A: Hi β,
Unfortunately the backup was corrupted and at present I cannot run again the experiments to obtain that data. α

BENCHMARKS: COLORING, SPORTS AND UMPIRES

> I have always felt strongly that operations research needs more libraries of instances for various problem classes, along with listings of current best solutions.
> By tracking how well we solve problems over time, we can show how we advance as a field. It also makes it easier to evaluate new work, making both authors and referees work easier.

Micheal Trick's OR Blog, October 17, 2011

PMS: MOTIVATIONS

- No more ``stand-alone'' papers
- Solution $\rightarrow$ Standard Problems:
  - Practical and with many real-world instances
  - Easy to understand, parse, validate,
  - Flexible: toward ``stand-not-alone'' paper
  - Issues:
    - Which and how many standards?
    - Who sets the standards?
- We need: (Web-Based) Problem Management Systems!
  - Problem management
  - Data management
  - Automatic validation of input and output
- An early attempt in (Bonutti et al. 2012) for the Curriculum-Based Course Timetabling Problem

PMS: MOTIVATIONS

- No more ``stand-alone'' papers
- Solution $\rightarrow$ Standard Problems:
  - Practical and with many real-world instances
  - Easy to understand, parse, validate,
  - Flexible: toward ``stand-not-alone'' paper
  - Issues:
    - Which and how many standards?
    - Who sets the standards?
- We need: (Web-Based) Problem Management Systems!
  - Problem management
  - Data management
  - Automatic validation of input and output
- An early attempt in (Bonutti et al. 2012) for the Curriculum-Based Course Timetabling Problem

WEB-BASED PROBLEM MANAGEMENT SYSTEM

- Validation of solutions, online and offline (open-source)
- Statistics on the instances
- Visualization of solutions
- Insertion of
  - Solutions (upon automatic validation)
  - Lower bounds
  - Instances (upon automatic validation and human check)
- Ensures data retention

Other approaches to reproducibility

THE RECOMPUTATION.ORG INITIATIVE

T11: Replication and Recomputation in Scientific Experiments
(Tuesday, August 19, 11:00--12:30)
Ian Gent and Lars Kotthoff

1. Computational experiments should be recomputable for all time
2. Recomputation of recomputable experiments should be very easy
3. It should be easier to make experiments recomputable than not to
4. Tools and repositories can help recomputation become standard
5. The only way to ensure recomputability is to provide virtual machines

PROGRAMMING BY OPTIMIZATION
(H. Hoos 2012)

Keynote Lecture

Holger H. Hoos, Machine Learning & Optimisation: Promise and Power of Data-driven, Automated Algorithm Design
Thursday, August 21, 9:00-10:00

PbO in a nutshell

- The specification of large and rich combinatorial design spaces of programs that solve a given problem, by means of avoiding premature commitment to certain design choices and active development of promising alternatives for parts of the design.
- The automated generation of programs that perform well in a given use context from this specification, by means of optimisation techniques that can realise the performance potential inherent in the given design space.

Levels of PbO:

- Level 0: Optimize settings of parameters exposed by existing software (parameter tuning)
- Level 1: Expose design choices hardwired into existing code (magic constants, hidden parameters, abandoned design alternatives)
- Level 2: Keep and expose design choices considered during software development
- Level 3: Strive to provide design choices and alternatives
- Level 4: No design choice that cannot be justified compellingly is made prematurely

Managing Experiments

THE CONTEXT

A significant portion of our time working in optimization is spent

- designing experiments,
- running experiments,
- interpreting experiments results.

A COMMON APPROACH

Often, this is accomplished through collection of throw-away scripts

- a variety of languages (bash, Perl, batch, you name it),
- parameters tuned manually (or almost),
- results saved into files with sophisticated naming conventions,
- manually patched and re-run when bugs in the code are discovered.

PROBLEMS

This approach has many drawbacks

- scripts become bloated very quickly,
- difficult to keep track of parameter setups,
- difficult to keep track of code versions,
- in time, results become difficult to retrieve.

Results: lost data, hindered repeatibility, lot of time wasted in mechanical tasks.

A SYSTEMATIC APPROACH

We advocate a more systematic approach to experiment management. The goals

- simplify the *design* of batches of experiments,
- standardize the *encoding* of batches of experiments,
- automate experiments *running* and *retrieval*.

On top of that: proper *result persistence*, automatic *parameter tuning*, *parallel execution* of experiments.

A TOOL: JSON2RUN

A *generic* command line tool (`j2r`) for experiment design, execution and analysis

- compatible with *any* executable,
- *batches* represented as *parameter trees* (see examples),
- automatic *parameter tuning* through F-Race,
- *parallel execution* of experiments + *stop* and *resume*,
- *persistence* on database,
- facilities to simplify the *analysis* of results.

TECHNOLOGIES

- *JSON*, to concisely represent experiment batches,
- *MongoDB*, schema-less database to store results (JSON-based),
- *R*, language for statistical analysis of results,
- *Python*, glue language to stick everything together.

Overall, tiny utility: ~1300 lines of Python + ~300 lines of R. Manual included.

JSON

Compact data interchange format (*J*ava*S*cript *O*bject *N*otation) captures common concepts in data structures

- scalars: *numbers* (floats, ints), *strings*, *booleans*, *null*,
- *arrays* (actually lists, as elements can be heterogeneous),
- *objects* key-value collections

BATCH REPRESENTATION

Basic usage, *json2run* as command generator from tree-like structure with 2 ***leaf*** types

- *discrete*, parameters with explicit values, e.g., 1, true, ``sa'', 3.14,
- *continuous*, parameters with values defined by intervals, e.g, $[0, 1]$

and 2 ***inner*** types

- *and*, combination of parameters (Cartesian product),
- *or*, alternative parameters.

Plus *post-processors* to manipulate their results.

BATCH REPRESENTATION

Two syntaxes

- *standard syntax* more verbose, tested
- *compact syntax*
  - much shorter (see examples)
  - *experimental*

Both syntaxes are valid JSON.

STANDARD REPRESENTATION

```
{
    "type": "and",
    "descendants": [
        {
            "name": "heuristic",
            "type": "discrete",
            "values": [ "ts", "sa" ]
        },
        {
            "name": "iterations",
            "type": "discrete",
            "values": { "min": 100, "max": 1000, "step": 100 }
        }
    ]
}
```

COMPACT REPRESENTATION

```
{
    "and": [
        {
            "heuristic": [ "ts", "sa" ]
        },
        {
            "iterations": { "min": 100, "max": 1000, "step": 100 }
        }
    ]
}
```

OUTPUT

Executable specified with `-e, --executable`

```
$ j2r -i test.json -e ./solver
./solver --heuristic ts --iterations 100.0
./solver --heuristic ts --iterations 200.0
...
./solver --heuristic ts --iterations 1000.0
./solver --heuristic sa --iterations 100.0
./solver --heuristic sa --iterations 200.0
...
./solver --heuristic sa --iterations 1000.0
```

ARBITRARILY COMPLEX ***PARAMETER TREES***

```
{
    "or": [
        {
            "and": [
                { "heuristic": [ "ts" ] },
                { "tabu_s": [ 10, 15, 20 ] }
            ]
        },
        {
            "and": [
                { "heuristic": [ "sa" ] },
                { "t_init": { "min": 50, "max": 100, "step": 10 } }
            ]
        }
    ]
}
```

Output

```
./solver --heuristic ts --tabu_s 10
./solver --heuristic ts --tabu_s 15
./solver --heuristic ts --tabu_s 20
./solver --heuristic sa --t_init 50.0
./solver --heuristic sa --t_init 60.0
./solver --heuristic sa --t_init 70.0
./solver --heuristic sa --t_init 80.0
./solver --heuristic sa --t_init 90.0
./solver --heuristic sa --t_init 100.0
```

FILE AND DIRECTORY PARAMETERS

It is often the case that one of the parameters is the path of a problem instance, somewhere on the drive. The `file` and `directory` parameters are special `discrete` parameters whose values are generated from reading files or directories.

In case of the `file`, all the lines of the file matching the regular expression in `match` become values of the parameter.

In case of the `directory`, all the file names matching the regular expression in `match` become values of the parameter.

POST-PROCESSORS

Post-processors are used to manipulate the output of inner and leaf nodes, and provide services such as

- rounding parameter values,
- generating new parameters (discrete or continuous) from
  - the output of other nodes, and
  - Python expressions
- sorting the order of parameters in the commands,
- providing an unique index to the configuration.

More post-processors described in the manual.

POST-PROCESSORS

In order to be used, they must be attached to an inner node

```
{
    "type": "and",
    "descendants": [ ... ],
    "postprocessors": [
        {
            "type": "rounding",
            "round": {
                "start_temp": 2
            }
        }
    ]
}
```

post-processors attached to a node are activated in order.

POST-PROCESSORS, COMPACT SYNTAX

In compact syntax, there is no `postprocessors` field in inner nodes, postprocessors *are* special nodes, with an *on* field, e.g.

```
{
    "rounding": {
        "start_temp": 2
    },
    "on": {
        "and": [ ... ]
    }
}
```

post-processors can be chained arbitrarily.

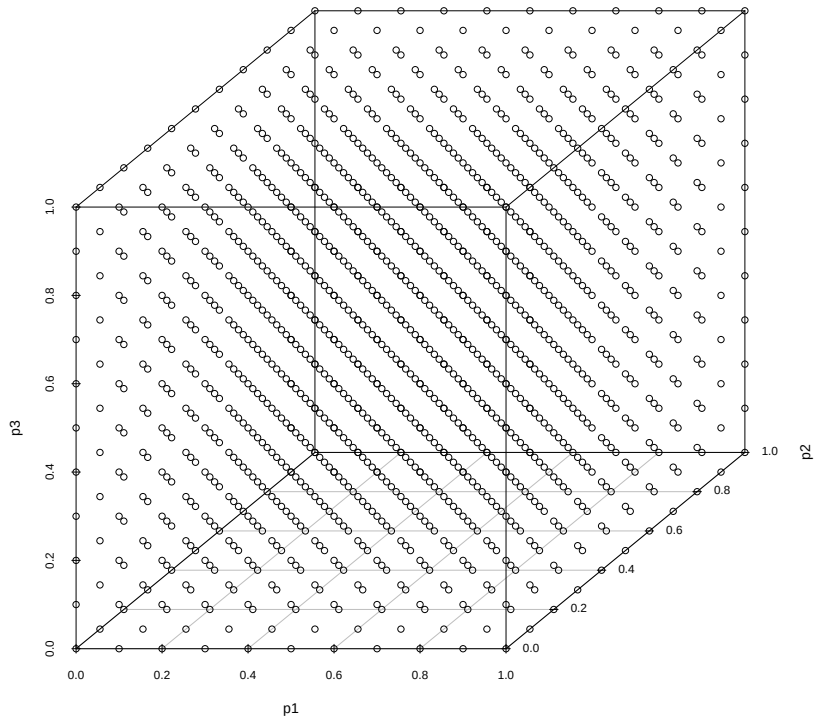
CONTINUOUS PARAMETERS

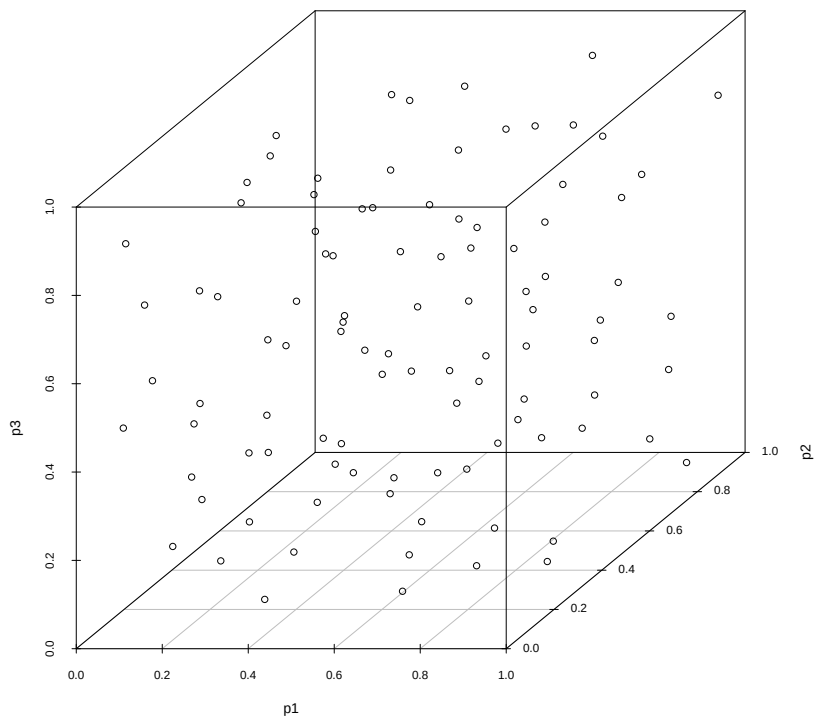
Continuous parameters must be sampled by a *post-processor* which defines how numbers must be chosen within the interval.

The default (and, so far, only) sampler in *json2run* generates n parameter configurations from the k-dimensional Hammersley point set, where k is the number of continuous parameters.

Scalable, random-like, space-filling.

HAMMERSLEY VS. FULL-FACTORIAL





HAMMERSLEY

Example, *continuous*, sampling, rounding

```
{
    "hammersley": 10,
    "on": {
        "and": [
            { "learning_rate": { "min": 0.1, "max": 0.3 } },
            { "lambda": { "min": 3.0, "max": 10 } }
        ],
    }
}
```

OUTPUT

```
./solver --learning_rate 0.12 --lambda 6.5
./solver --learning_rate 0.14 --lambda 4.75
...
./solver --learning_rate 0.3 --lambda 5.1875
```

Continuous and *discrete* parameters can be combined (e.g., `and`).

RUNNING EXPERIMENTS

If the executable outputs valid JSON on the standard output, e.g.

```
$ ./solver ...
{
    "cost": <solution_cost>,
    "time": <time_taken>,
    "solution": "..."
}
```

json2run can take care of the batch execution (`-a, --action`, `-n, --batch-name`)

```
$ j2r -a run-batch -i test.json -e ./solver -n <batch_name>
```

RUNNING EXPERIMENTS

There is virtually *no limit* on the information that can be printed by executable, as long as it is *valid JSON*.

All the output is saved on the database, along with the parameters used to run each experiments, and a whole lot of meta-information.

By default, *json2run* takes all the available cores on the host machine and schedules *one experiment for each core*, a different number of parallel experiments can be specified with `-p, --parallel-threads`.

RUNNING EXPERIMENTS

A running batch looks like this

```
08/11/2014 12:17:07 AM: Running batch with 4 parallel threads and non
    greedy.
08/11/2014 12:17:07 AM: Initializing workers ...
08/11/2014 12:17:07 AM: Generating experiments ...
08/11/2014 12:17:07 AM: Running (2/10) ./solver --learning_rate 0.14 --
    lambda 4.75
08/11/2014 12:17:07 AM: Running (1/10) ./solver --learning_rate 0.12 --
    lambda 6.5
...
08/11/2014 12:17:17 AM: Running (8/10) ./solver --learning_rate 0.26 --
    lambda 3.44
```

REPETITIONS

When dealin with *stochastic* solvers, it is often necessary to run a certain *repetitions of the same experiments*, in order to have a robust estimate of the solver's performance, e.g., median or mean.

Repetitions are supported by *json2run* through the `-r, --repetitions` parameter. If a batch generates n configurations, and $r = 30$, then the total number of experiments is $30n$.

RUNNING EXPERIMENTS

The batch `batch_name` is used to retrieve the set of experiments (parameters and results) from the local `j2r` database where they are saved (unless a different database is specified on the command line with `-dh, --database-host`).

Along with parameters and results, more meta-information is saved

- the user running the batch,
- the name of the host machine,
- the content of `batch.json` (after processing),
- the code revision (if parameter `--scm [git,hg]` is used),
- starting and ending times
-

PARAMETER TUNING

Parameter tuning is supported natively by *json2run*. The process is similar to running a normal batch, except for two additional parameters

- `-ip` instance parameter (the name of the parameter representing the instance),
- `-pp` performance parameter (the name of the parameter representing the solution cost).

All the generated parameter configurations are considered candidates.

PARAMETER TUNING

The parameter tuning procedure implemented in *json2run* is ***F-Race***, it starts by running each parameter configuration on an initial block of instances (`--initial-block`), then iteratively removes the configurations that are *significantly* inferior, from a statistical standpoint.

The confidence level of the F-Race can be set by providing a parameter `--confidence` to *json2run*.

MONITORING BATCHES AND RACES

It is possible to inspect the stored and running batches by running

```
$ j2r -a list-batches
Batches matching criteria: 2
Name | Completion | Host | User | Type | Started | Finished | ETA |
     Active
--------------+-----------+------+------+------+---------------+----------

my_other_batch | 10.00 % | red | turli | full | 10/08/14 14:39 | never |
      -- | *
my_batch | 100.00 % | red | turli | full | 10/08/14 14:17 | 10/08/14
     14:17 | -- |
```

running batches are marked with an `*` under the *Active* column. The *Completion* and *ETA* columns give an estimate of the time to completion.

The list of batches can be filtered by providing a `-f, --filter` parameter.

GETTING BATCH INFO

In order to get the complete information about a batch or a race, you can run

```
$ j2r -a batch-info -n <batch_name>
```

For races, it is possible to query the best configuration so far, or all the non-inferior configurations with

```
$ j2r -a show-best -n <race_name>
$ j2r -a show-winning -n <race_name>
```

Output is always in JSON.

INTERRUPTING EXPERIMENTS

Batches and races can be interrupted at any time, by terminating the main `j2r` script (kills are signaled upon termination).

In order to resume a previously interrupted batch, it is sufficient to run

```
$ j2r -a run-batch -n <batch_name>
```

or

```
$ j2r -a run-race -n <race_name>
```

in case of a race. The previous settings used to run a batch or race will be used.

RETRIEVING EXPERIMENTS

The results can be retrieved in good ol' CSV format with the command

```
$ j2r -a dump-experiments -n <batch_name> >> experiments.csv
```

however it is often convenient to do your analysis in *R*, by including the `analysis.R` script that comes with the *json2run* distribution and retrieving the batch in data frame format

```
> source("analysis.R")
> connect("localhost")
> x <- getExperiments("<batch_name>")
```

RESOURCES

The *json2run* project is open-source (premissive MIT License)

- Bitbucket **https://bitbucket.org/tunnuz/json2run**
- GitHub **https://github.com/tunnuz/json2run** (mirror)

The `README.md` provides an extensive documentation for installing *json2run*, and of all the basic and advanced functionalities, along with examples.

If you use it, you might consider citing the tech report (not mandatory) **http://arxiv.org/abs/1305.1112**.

TEAMWORK

TEAMWORK

Research projects are usually collaborative works

- Code developed with co-workers
- papers written with co-authors

It is important to use the right tools in order to make collaboration straightforward.

CODE VERSIONING

Code versioning if maybe the single most relevant tool for collaboration in optimization research

- Allows code to ***go back in time***, good for trying out alternatives
- a principled backup metod,
- Can be used for both code and papers
- Automatic merging makes it easy to work on different parts of the project, or paper

CODE VERSIONING

Centralized code versioning systems (CVS) - one (master) *repository*, multiple (slave) *working copies*, - typically requires to be *connected* to the network, - changes on a working copy are ***committed*** to the repository, - working copies can get ***updated*** to the current status of the repository.

Examples: cvs, Subversion

CODE VERSIONING

Distributed code versioning systems (DCVS) - many equivalent *clones* of the original repository, - each repository is *independent*, no need to be connected, - changes can be locally ***committed*** - local versioning does not affect everyone's code, - still allows going back in time, - tested changes can be ***pushed*** to the original repository,

Examples: git, Mercurial

CODE VERSIONING, CON'D

In general, DCVS are more convenient

- plenty of disk space, so complete *clones* are *not an issue*,
- allow versioning in *mobility* (good for traveling people),
- automatic *merges are easier* (because of more frequent commits)

Plenty of hosting services with academic licenses, e.g., GitHub, Bitbucket.

RESOURCES

- Git
  - Doc http://git-scm.com/documentation
- Mercurial
  - Hg Init http://hginit.com/
  - Doc http://mercurial.selenic.com/guide

OTHER COLLABORATION TOOLS

Other tools that facilitate collaborations are

- Communication
  - *conference calls*, e.g., Skype, Hangouts, etc.
  - *e-mail/mailing lists*, useful to keep track of messages,
- collaborative *task management*,
  - e.g., Basecamp, Trello.
- *code-related*,
  - bug trackers, often included in hosted services such as GitHub, Bitbucket.

ABOUT HOSTED SERVICES

Hosted services are a *valuable resource*

- often offer an *ecosystem of tools*,
  - e.g., repositories, wikis, etc.,
- most of them offer *academic licensing*,
- researchers can *focus on research*, rather than administrative tasks.

However, might not be compatible with local policies, e.g., about hosting code on external services. Check with your department.

Take home messages

TAKE HOME MESSAGES

- Software engineering is *not only for industry*
- many benefits for research as well
  - improved reproducibility (hot topic) and maintainability
  - faster prototyping w/ modular approach
  - design reuse w/ software frameworks
- Overall
  - less effort spent ***making stuff work***
  - more time for ***interesting research***
  - and, realiability also means *less stress* :)

Thank you for your attention!

SOFTWARE ENGINEERING FOR SEARCH AND OPTIMIZATION PROBLEMS

Tutorial at ECAI-2014

August, 18th 2014

Luca Di Gaspero Tommaso Urli

SaTT, Università degli Studi di Udine, Italy

ORG, NICTA, Canberra, Australia

LICENSING

One of the many aspects to consider when developing software for research, is licensing. Licensing affects

- how our code can be used by other people, and
- how we can other people's code.

Mainly relevant if *your code is* open source, or if you *make use of* open source code.

PRINCIPLES OF COPYRIGHT LAW

In absence of a different license, the creator of a *work* (be it a poem, a painting, a piece of code) is the only person who has the *right* of copy, display, create derivative works, or make commercial use of it.

This right has a fairly long lifetime (usually past the life of the creator).

A copyright does not need to be made official in order to be in effect.

PRINCIPLES OF COPYRIGHT LAW

However,

> ***Copyright law does not protect any particular idea. Rather, copyright protects only the expression of that idea.***

and

> ***This limitation to expressions excludes protection from copyright of creations that are not expressed in a tangible, reproducible medium.***
>
> ***This limitation to the expressions of an idea is the principal distinction between the applications of patent and copyright.***

PRINCIPLES OF COPYRIGHT LAW

Exceptions

- *Work for hire* works made by an employee in the scope of its employment, or works produced as commissioned by someone else, are still protected by the copyright law, but the rights belong to the employer of the creator.
- *Fair use* certain uses of a work are allowed, as long as they do not hinder the commercial exploitation of the work by the creator (subject to interpretations).

OPEN SOURCE

Open source licenses often stick to the following principles

1. *Open distribution* allow a user to redistribute (for free or under compensation) the software, and
2. *open modification* allow a user to modify the software.

Some licenses may require derivative work to be licensed under the same principles (*generational limitation*).

SOME LICENSES

We briefly review some of the open source licenses that can be used on research code, pointing out

- how much *permissive* they are,
- whether they hold a *generational limitation*,
- if there are any additional limitations.

MIT LICENSE

The MIT license in general one of the most permissive licenses, it allows to deal with the work

> ***without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies [...] and to permit persons [...] to do so***

as long as the license notice is included in all significant portions of the work.

Similar licenses BSD, Apache (forbids endorsement of derivatives by the creator of the work)

ACADEMIC FREE LICENSE

Similar to the Apache License, in forbidding endorsement of derivatives by the creator. Adds more clauses related to the open source licensing in case the work has a patent on it.

Fairly more complex than the others. May be of interest if some of the research work you carry out has a patent on it, but you still want to open source it.

GNU GENERAL PUBLIC LICENSE (GPL)

Main difference with the previous, has a *generational limitation*.

Redistribution is allowed, as the creation of derivative works, as long as the derivative work is licensed under the GPL license.

Also, the work must be free of patents.

GNU LESSER GENERAL PUBLIC LICENSE

Less restrictive version of the GPL. Created with the aim of allowing linking against libraries not licesed under the GPL.

Allows the redistribution of the licensed code, and the combination with other code which is not licensed in the same way.

Again, fairly complex with respect to MIT or BSD licenses.

BOTTOM LINE

Bottom line: in case you consider open source licensing, check with your department / company what is the policy on the matter.

Most departments have established (or almost) rules on how to license work made under employment.

REFERENCES

A very interesting and accessible reading on open source licensing is ***Understanding Open Source Licensing*** by Andrew M. St. Laurent (O'Reilly, 2004).

History of licensing, descriptions of most open source licenses in layman terms (also, Creative Commons, Mozilla licenses, etc.).

Now freely available at
http://oreilly.com/openbook/osfreesoft/book.