A modeling-language based approach to automatically recommend optimization methods

Sofiane Tanji

ICTEAM/INMA
Université catholique de Louvain

Joint work with François Glineur (UCLouvain)

INFORMS Annual Meeting 2024 | Sunday, October 20, 2024

Motivating example: LASSO regression

Consider $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$, $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_+^*$. LASSO regression solves:

$$\hat{\beta} \in \arg\min_{\beta \in \mathbb{R}^d} \left\{ \frac{1}{2} \|y - X\beta\|^2 + \lambda_1 \|\beta\|_1 \right\}, \qquad \text{(nonsmooth convex)}$$

or equivalently,

$$\hat{\beta} \in \arg\min_{\beta \in \mathbb{R}^d} \left\{ \frac{1}{2} \|y - X\beta\|^2 \text{ subject to } \|\beta\|_1 \leq \lambda_2 \right\} \qquad \text{(QP)}$$

or equivalently,

$$\hat{\beta} \in \arg\min_{\beta \in \mathbb{R}^d} \left\{ \|\beta\|_1 \text{ subject to } \|y - X\beta\|^2 \leq \lambda_3 \right\} \qquad \text{(SOCP)}$$

For each formulation, you can use many different methods:

Formulation	Some applicable methods
nonsmooth convex	coordinate descent, subgradient method, proximal gradient ...
convex QPs	splitting methods, active-set methods
SOCPs	augmented lagrangian, splitting methods

Four steps approach

S0. *Gather* methods with associated complexity results

S1. *Match* a given formulation with all applicable methods

S2. *Reformulate* a given problem to find equivalent formulations

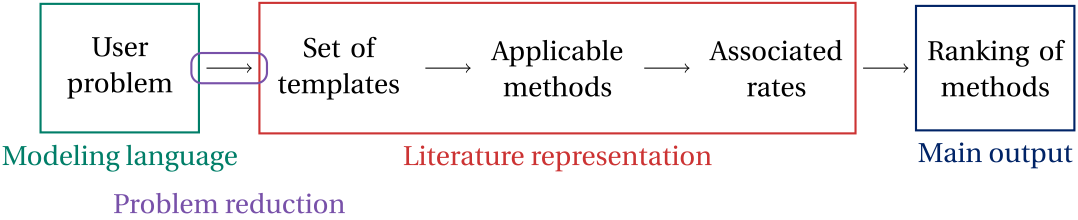
S3. *Compare* complexity results of (formulation, method) combinations

In this talk

We present the Optimization Methods Ranking Assistant (OMRA)

- A repository of convergence results existing in the literature
- A modeling language to describe optimization problems
- Detection of methods applicable to user-provided problems
- Comparison of applicable methods by worst-case performance
- all of this framed in a PYTHON toolbox.

Pipeline



Outline

Four steps approach

S0. ***Gather* methods with associated complexity results**

S1. *Match* a given formulation with all applicable methods

S2. *Reformulate* a given problem to find equivalent formulations

S3. *Compare* complexity results of (formulation, method) combinations

Setting : What optimization problems usually look like

Consider any optimization problem in the black-box form:

$$\min_x f(x) \tag{1}$$

where f is decomposed into simpler components f_i, for example:

$$f(x) = f_1(x) + f_2 \circ f_3(x) + \sum_{j=1}^{n} f_4^j(x) - \max_{j=1,\dots,n} f_5^j(x) \tag{2}$$

with

- possible assumptions on the f_i's (convexity, Lipschitz continuity, etc.)
- f_i can also correspond to "atom functions": ℓ_1-norm . . .
- access to certain oracles for each f_i (subgradient, proximal operator, etc.)

Example of what you can find in the literature

Theorem: Worst-case convergence rate of Fast Proximal Gradient Method

Take $f : \mathbb{R}^n \to \mathbb{R}$ L-smooth and convex and $h : \mathbb{R}^m \to \mathbb{R}$ closed, proper, convex. Suppose $\|x_0 - x_*\| \leq R$.
Then, the Fast Proximal Gradient Method with stepsize $\frac{1}{L}$ applied to minimizing $F(x) = f(x) + h(x)$ satisfies for all $n \geq 1$:

$$F(x_n) - F(x_*) \leq \frac{2L}{n^2 + 5n + 2} \|x_0 - x_*\|^2 \tag{3}$$

Complexity results follow the same skeleton

Theorem: Worst-case convergence rate of Algorithm 1

Suppose assumptions $\{(A1), (A2), (A3)\} = \mathcal{A}$ hold.
Consider some initial conditions $\mathcal{I}$.
Then, Algorithm 1 with parameters $\mathcal{P}$ applied to Problem (1) satisfies for all $k \geq 1$

$$F(x_k) - F(x^*) \leq \varphi(k, \mathcal{A}, \mathcal{I}, \mathcal{P}) \tag{4}$$

Existing results (always) have

- Some template optimization problem (in red)
- The considered optimization method (in purple)
- A rate of convergence (in green)

Elements to encode known complexity results

Template problem

- ✔ This is just an optimization problem (use a modeling language)

Optimization method

- Enough to encode the parameters.
- Dependence of method parameters to template parameters

Convergence rate

Example: Convergence rate of FISTA with stepsize $1/L$

$$\underbrace{F(x_N) - F_*}_{\text{performance measure}} \leq \frac{2L}{n^2 + 5n + 2} \overbrace{\|x_0 - x^*\|^2}^{\text{initial conditions}}$$

with template parameter L and method parameter $1/L$.

Outline

Four steps approach

S0. *Gather* methods with associated complexity results

S1. *Match* a given formulation with all applicable methods

S2. *Reformulate* a given problem to find equivalent formulations

S3. *Compare* complexity results of (formulation, method) combinations

How to write an optimization problem in OMRA

Problem

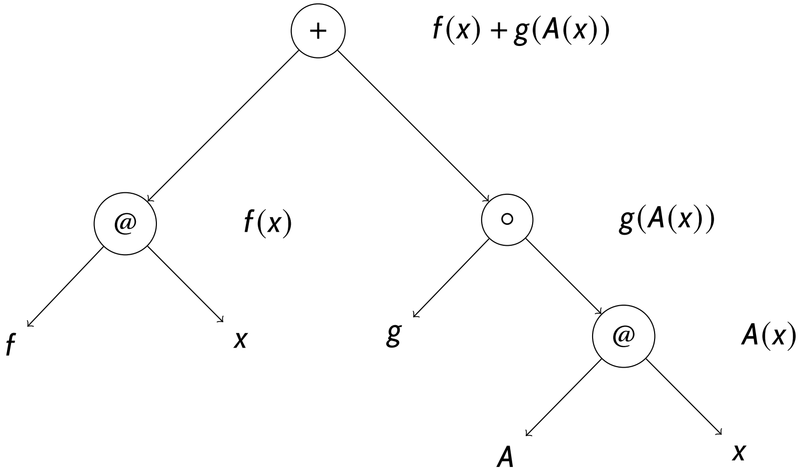
$$\min_{x \in \mathbb{R}^n} f(x) + g(A(x)) \qquad (5)$$

where

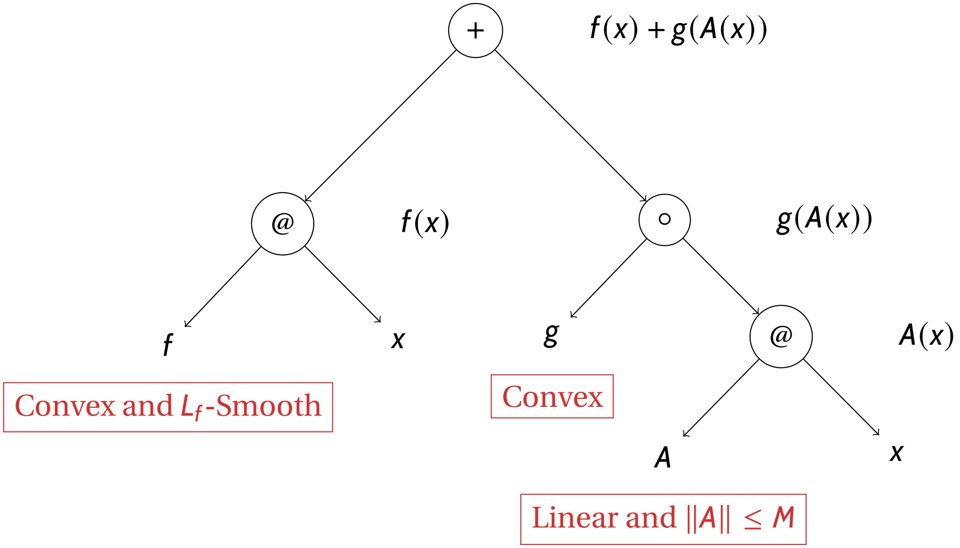
- $f : \mathbb{R}^n \to \mathbb{R}$ is convex, L_f-smooth. Access to ∇f.
- $g : \mathbb{R}^m \to \mathbb{R}$ is convex. Access to $prox_g$.
- $A : \mathbb{R}^n \to \mathbb{R}^m$ is a linear mapping with $\|A\| \leq M$. Access to $x \to Ax$.

```
pb = Problem()
x = pb.declare_variable("x", Rn)
f = pb.declare_function("f", Rn, R)
g = pb.declare_function("g", Rm, R)
A = pb.declare_function("A", Rn, Rm)
pb.set_objective(f(x) + g(A(x)))
f.add_property(Convex())
f.add_property(Smooth(0, 10.))
g.add_property(Convex())
A.add_property(Linear(10.))
pb.declare_oracle(Derivative(f))
pb.declare_oracle(Proximal(g))
pb.declare_oracle(Evaluation(A))
```

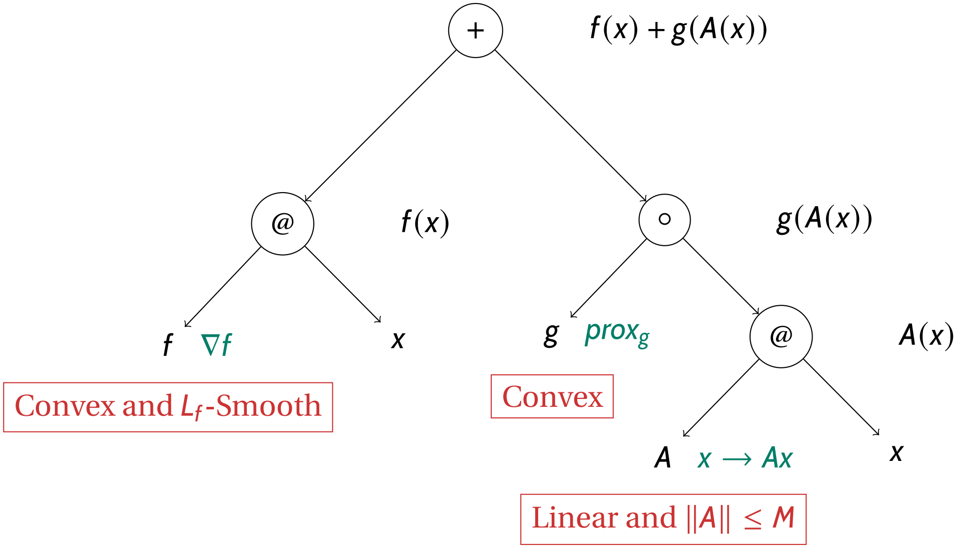
Optimization problem as a Directed Acyclic Graph



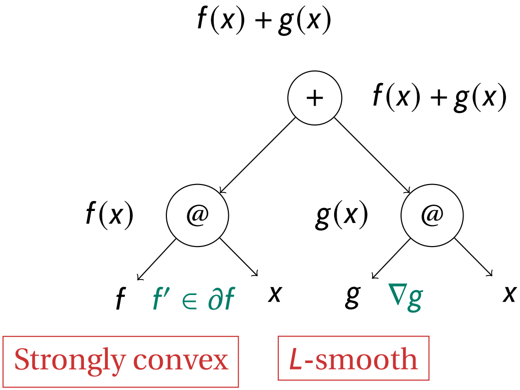
Optimization problem as a Directed Acyclic Graph



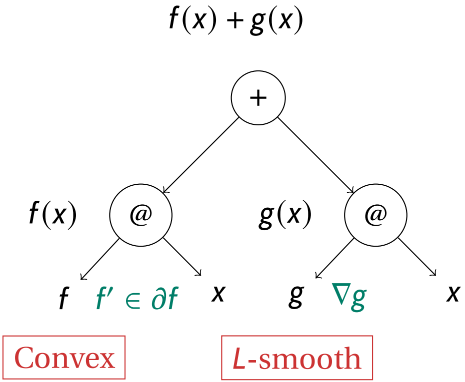
Optimization problem as a Directed Acyclic Graph



Finding applicable methods means matching the problem to a template

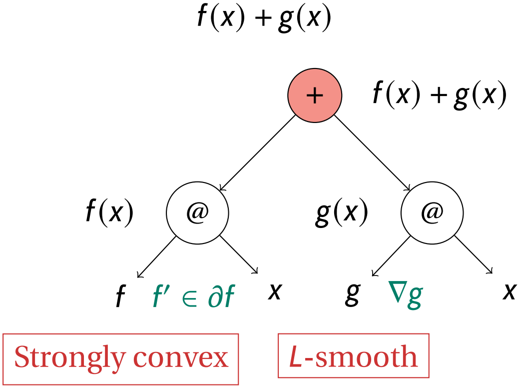


Optimization Problem P

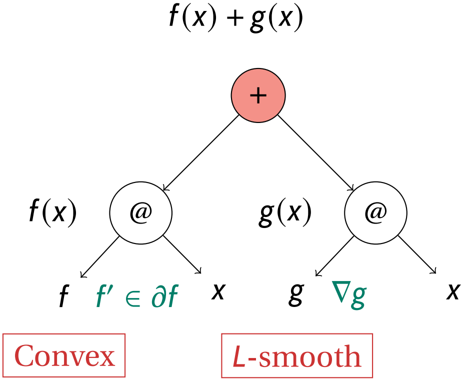


Optimization Template T

Finding applicable methods means matching the problem to a template

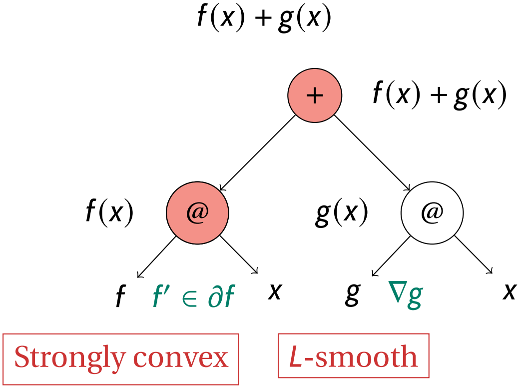


Optimization Problem P

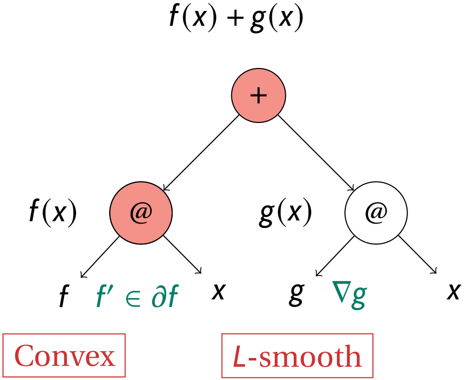


Optimization Template T

Finding applicable methods means matching the problem to a template

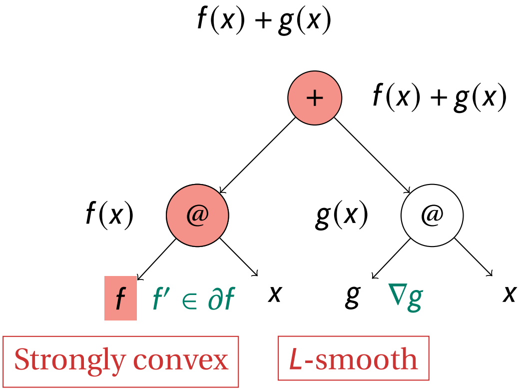


Optimization Problem P

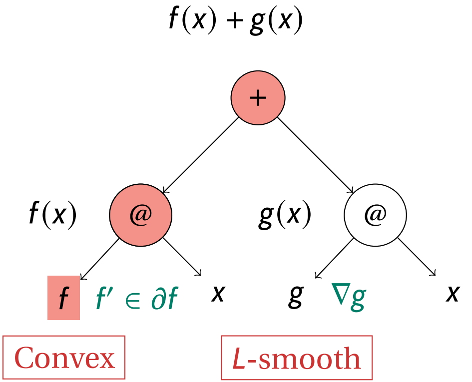


Optimization Template T

Finding applicable methods means matching the problem to a template

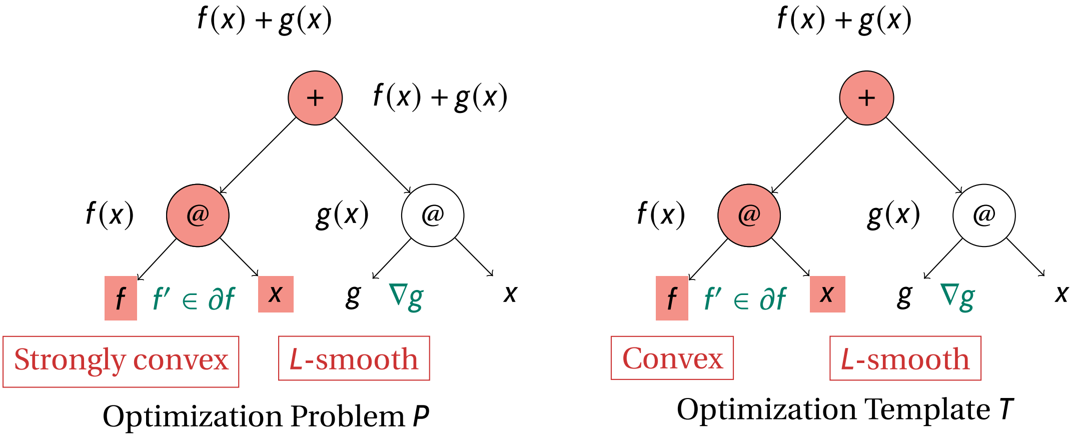


Optimization Problem P

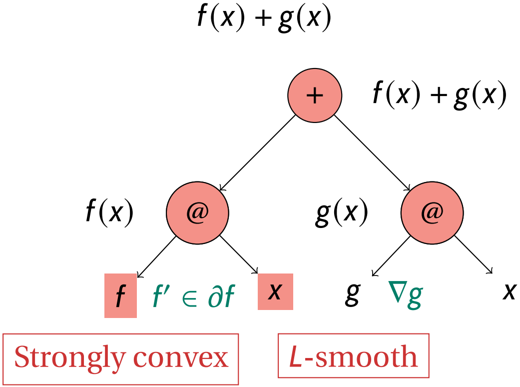


Optimization Template T

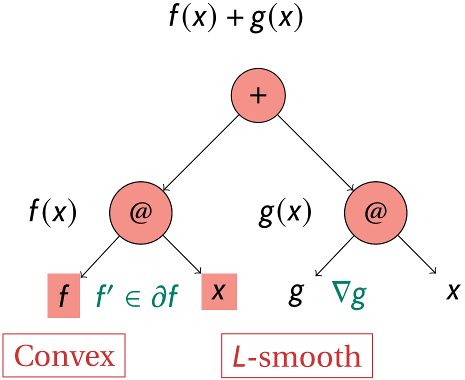
Finding applicable methods means matching the problem to a template



Finding applicable methods means matching the problem to a template

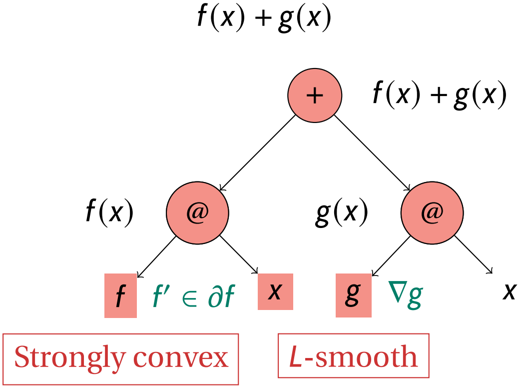


Optimization Problem P

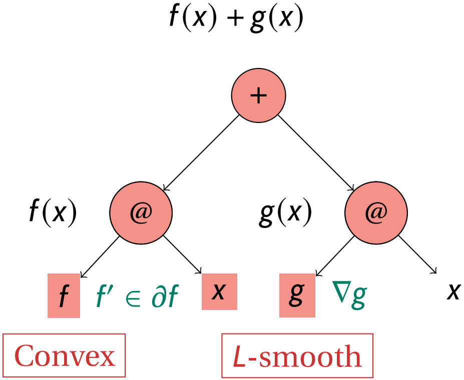


Optimization Template T

Finding applicable methods means matching the problem to a template

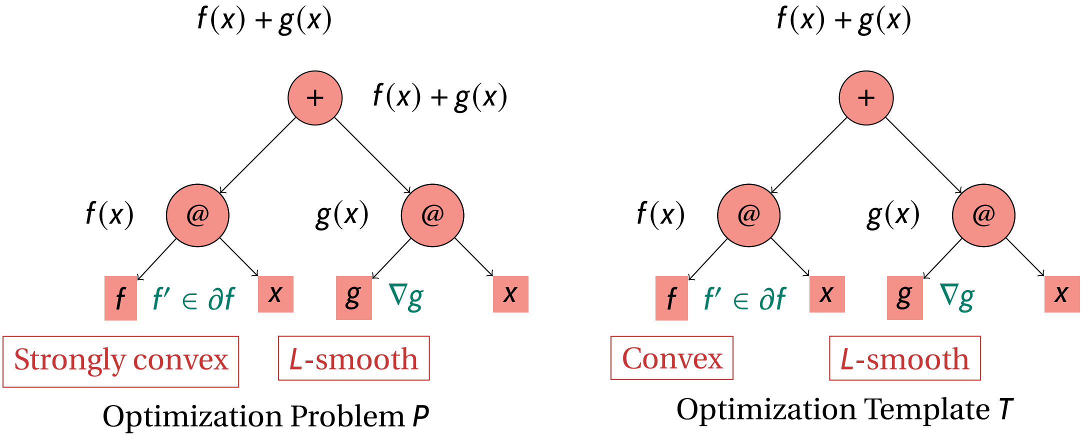


Optimization Problem P



Optimization Template T

Finding applicable methods means matching the problem to a template



Outline

Four steps approach

S0. *Gather* methods with associated complexity results

S1. *Match* a given formulation with all applicable methods

S2. ***Reformulate* a given problem to find equivalent formulations**

S3. *Compare* complexity results of (formulation, method) combinations

Scenario 2 : reformulations

We need to match user-provided problems to templates.

- ✔ Ideal scenario is immediate match of user-provided problem to template.
- Solution otherwise: use *mathematical results and reformulation tricks !*

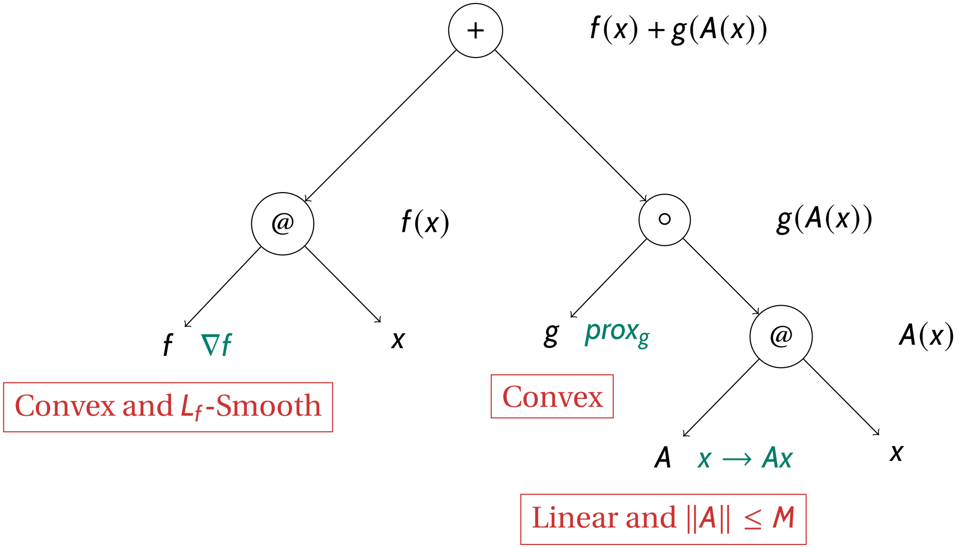
Mathematical results:

- Sum of smooth (resp. convex) functions is smooth (resp. convex),
- Proximal operator of $f + \gamma\|x\|^2$ is computable given $prox_f$ ...

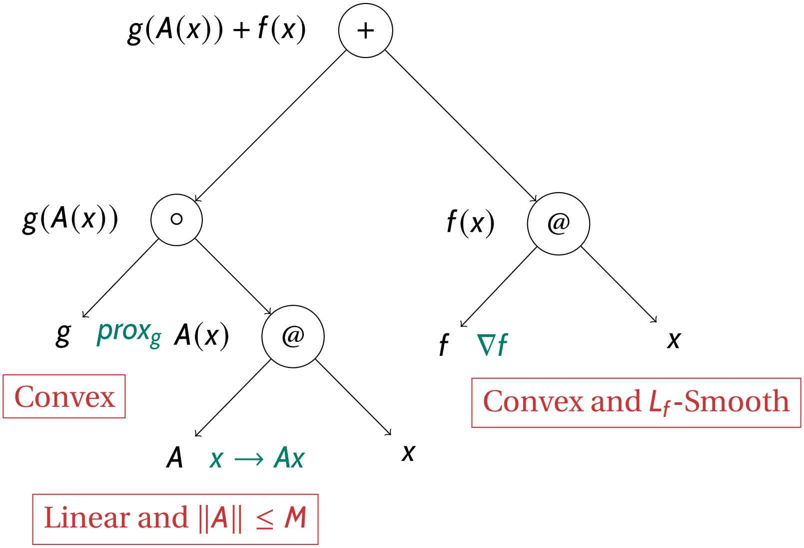
Reformulation tricks:

- Commutativity of operators,
- losing structure and regrouping terms,
- transfer of curvature (parametrized reformulation),
- computing oracles as a subproblem (parametrized reformulation) ...

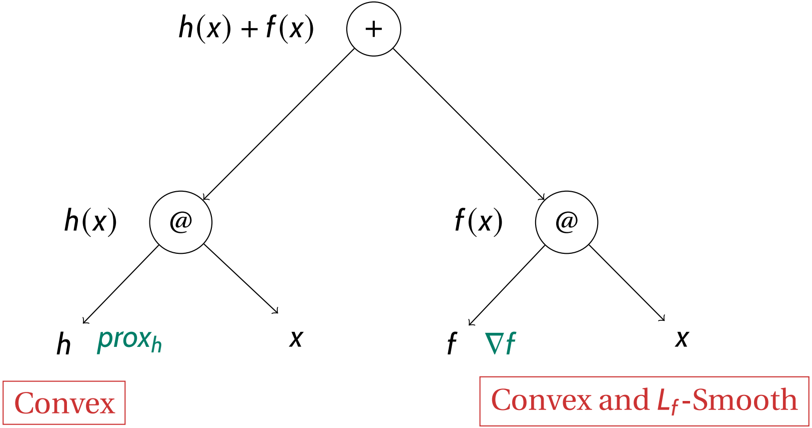
Reformulation example



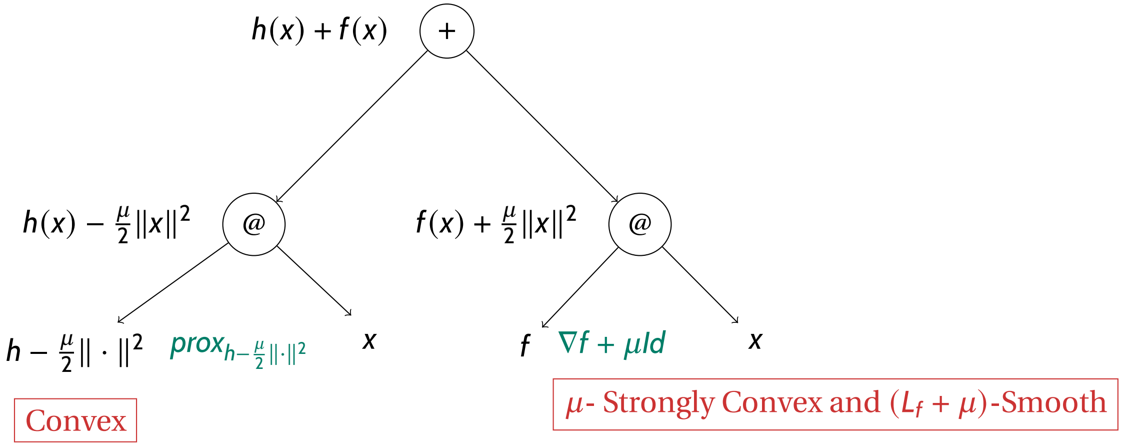
Commutativity of the sum *operator*



Losing structure



Transfer of curvature



Outline

So far: from a user-provided problem, we get a list of (Template, Method, Rate)

S0. *Gather* methods with associated complexity results

S1. *Match* a given formulation with all applicable methods

S2. *Reformulate* a given problem to find equivalent formulations

S3. ***Compare* complexity results of (formulation, method) combinations**

Our ranking criterion:
The convergence rate associated to each (template, method) combination *(× the computational cost per iteration)*

How to deal with sophisticated rate functions

Example: Convergence rate of fixed-step (γ) GD for f convex and L-smooth

$$f(x_N) - f_* \leq \frac{L}{2} \frac{\|x_0 - x^*\|^2}{1 + \gamma L \min\left\{2N, \frac{-1+(1-\gamma L)^{-2N}}{\gamma L}\right\}}$$

1. Compute rates numerically whenever possible
2. Drop asymptotically worse methods if high iteration budget
3. Compare leading coefficients whenever possible
4. *Sampling method*

Conclusions

Contribution: a principled approach to compare optimization methods and its Python implementation, OMRA

- Large repository of known results in the form (Template, Method, Convergence Rate)

Pipeline

User problem $\longrightarrow$ Set of templates $\longrightarrow$ Applicable methods $\longrightarrow$ Associated rates $\longrightarrow$ Ranking of methods

Modeling language

Literature representation

Main output

Problem reduction

Next steps

- *Make this encyclopedia available through a website* ***(very soon)***

Make the toolbox richer

- Aggregate more results in the database
- Add reformulation techniques (η-trick, duality (for the LASSO example))

User features

- Code generation for recommended methods

Thank you again for your attention!

Questions ?

Do not hesitate to contact me:

→ sofiane.tanji@uclouvain.be

→ https://sofianetanji.com

Preliminary results: Equivalent templates

Problem: Additive composite template

$$\min_{x\in\mathbb{R}^n} f(x)+g(x) \tag{6}$$

where

- $f:\mathbb{R}^n\to\mathbb{R}$ is μ_f-convex, L_f-smooth. Access to ∇f.
- $g:\mathbb{R}^n\to\mathbb{R}$ is μ_g-convex, L_g-smooth. Access to $prox_g$.

Problem: Difference of convex template

$$\min_{x\in\mathbb{R}^n} \varphi_1(x)-\varphi_2(x) \tag{7}$$

where

- $\varphi_1:\mathbb{R}^n\to\mathbb{R}$ is convex, L_{φ_1}-smooth. Access to $\nabla\varphi_1$.
- $\varphi_2:\mathbb{R}^n\to\mathbb{R}$ is convex. Access to $\nabla\varphi_2$.

Through the sequence of (implemented) transformations

$$
\begin{aligned}
f(x) + g(x) &= g(x) + f(x) \; \textit{(Commutativity of the sum operator)} && (8)\\
&= (g(x) + \gamma\|x\|^2) + (f(x) - \gamma\|x\|^2) \; \textit{(Transfer of curvature)} && (9)\\
&= (g(x) + \gamma\|x\|^2) - (\gamma\|x\|^2 - f(x)) \; \textit{(plus = minus minus)} && (10)\\
&= \varphi_1 - \varphi_2. && (11)
\end{aligned}
$$

- Properties: Depending on the parameter γ, we can make φ_1, φ_2 convex.
- Having $prox_g$ and ∇f, we can compute $\nabla\varphi_1, \nabla\varphi_2$.

Full matching algorithm

```
class Problem:
  [...]
  def compute_reformulations(self):
    visited = set()
    queue = [self.objective]
    while len(queue):
        current_tree = queue.pop(0)
        for g in TRANSFORMATIONS:
            for new_tree in current_tree.transform(g):
                if new_tree not in visited:
                    visited.add(new_tree)
                    queue.append(new_tree)
    return set(list(visited))
```