High-performance local search for a large-scale energy management problem

Frédéric Gardi **Karim Nouioua**

Bouygues e-lab, Paris
fgardi@bouygues.com

Laboratoire d'Informatique Fondamentale - CNRS UMR 6166,
Faculté des Sciences de Luminy - Université Aix-Marseille, Marseille
karim.nouioua@lif.univ-mrs.fr

EDF energy management problem

In a few words…

- Very large-scale mixed integer non linear problem (MINLP)
- Maybe the most challenging OR problem that we had to tackle

Proposed practical solution:

Local Search

But…

1) Pure: no decomposition, no hybridization, no metaheuristic.

2) Aggressive: millions of feasible solutions visited within the time limit.

Following the methodology by [*Estellon, Gardi, Nouioua, SLS 2009*], derived from successful past experiences (Renault, France Telecom, Air Liquide, Bouygues).

EDF energy management problem

Work focused on

1) Designing moves enabling an effective exploration of search space.

2) Speeding up the evaluation of moves.

Surrounded by an important effort in software engineering for ensuring the reliability: programming with assertions, checkers for incremental structures, continuous refactoring, CPU & memory profiling.

→ Critical in the quest of high performance

General heuristic

Heuristic decomposed into 3 phases:

Phase 1: Find an admissible scheduling of outages = respecting combinatorial constraints CT14-CT21.

Phase 2: Find a schedule with admissible production plan = spacing outages such that stocks do not exceed maximum levels before and after refueling operations (CT11).

Phase 3: Optimize the global cost of the admissible schedule.

Each phase tackled by local search : first-improvement descent with randomized selection of moves.

General heuristic

Phase 3: optimizing the global cost

How reducing the complexity induced by scenarios?

By working on a subset of scenarios (eventually aggregated).

The following strategy works well in practice:

A) Optimize on one scenario with average demands and T1 costs.
B) Refine the solution over all scenarios.

Step A is reinforced without losing efficiency: optimize on one scenario with average demands but with T1 completion costs computed over all scenarios.

Moves

Natural move:

Select *k* outages in the current solution and shift them over the time line. Size when no constraint: $O(H^k)$ with *H* the number of weeks.

Qualification stage: apply natural moves randomly with k = 1,2,3.

→ Very low success rate: premature rejection due to CT14-CT21

Idea: apply compound moves based on natural (small) moves, to reach feasible solutions with higher probability:

1) Apply a small move which may destroy feasibility.
2) Iterate small moves to repair violated combinatorial constraints.

Moves

Compound moves

→ Generalize ejection chains and destroy-repair methods
→ Jump from a feasible solution to another one by local search

Improvements:

- Select new starting dates respecting CT13 and CT11
- Target outages to destroy: random, constrained, consecutive
- Target outages to repair: inducing violations on CT14-CT21

→ 75 % of compound moves lead to new feasible solutions
→ Better convergence (speed, robustness)

Evaluation machinery

General evaluation scheme, for a subset *S* of scenarios:

Combinatorial part:

Perform small destroying move ;
While combinatorial violations remain do
    Perform small repairing move ;
If solution remain infeasible, then abort ;

Continuous part:

Set refueling amounts of impacted outages ;
For each scenario in *S* do
    Set production levels of impacted T2 plants ;
Compute global cost of new solution ;

Evaluation machinery

Combinatorial part

Violations on CT14-CT21 maintained by *O*(1)-time routines related to the arithmetic of intervals (union, intersection, inclusion, distance).

Minimum-Spacing Cut:

Spacing between consecutive outages *k* and *k+1* must be large enough to ensure CT11 at *k+1*, even if fuel reload at *k* is minimal and production on cycle *k* is maximal.

→ Strong combinatorial "cuts" induced by the continuous sub problem

Evaluation machinery

Combinatorial part

Minimum-Spacing Cut:

- Evaluated in $O(\log T')$ time by dichotomy in the worst case
- But in $O(1)$ amortized time by hash-map caching in practice

T': time steps between two consecutive outages

Since 80 % of the evaluation time is spent in the continuous part, then Minimum-Spacing Cut is crucial for efficiency.

Evaluation machinery

Continuous part

For impacted outage *k*, refueling amount is randomly set between:

- the minimum given in input
- the maximum to satisfy the minimum spacing to outage *k+1*

Evaluation machinery

Continuous part

For impacted production cycle *k*, production levels are computed using an *O*(*T'*)-time randomized-greedy algorithm:

1) Push production levels to maximum while imposition is not reached.

2) Compute analytically the maximum amount *m** of modulation.

3) Set modulation amount *m* randomly in [0, *m**].

4) Stock *s* to consume = stock after refueling – *m*.

5) Set production levels so as to consume stock *s*: either randomly from left to right, or driven by the lowest T1 completion costs.

Evaluation machinery

Continuous part

For each impacted T2 plant, T1 completion costs are computed over all scenarios in $O(\log(P1\ S))$ time using an extensive data structure.

Total time complexity: $O(P2'T'(S + \log(P1\ S)))$

P1 : T1 plants, *P2'* : impacted T2 plants,
T': impacted time steps, *S* : scenarios.

→ Almost linear in the size of changes on current solution
→ 10 000 times faster than linear programming (Gurobi)

Numerical experiments

Summary in numbers…

- Programmed in ISO C++: 10 000+ lines of code
- 15 % of code dedicated to checks (3 debug levels)
- Statically compiled with GCC 4 (-O3) on x86-64 platform
- *Gprof* for CPU profiling, *Valgrind* for memory profiling

- Continuous part treated in exact precision using 64-bits integers
- Low-level code optimization to reduce RAM footprint by 2
- 1 Go of RAM allocated for largest instances (B8, B9, B10)

- Fast convergence: 99 % of cost improvement in 10 mn
- 20 000+ compound moves per minute (100 000+ small moves)
- 1 000 000+ of feasible solutions explored per hour
- Improvement rate of compound moves: 1 %

Numerical experiments

Results obtained on 64-bits Linux with 2.93 GHz, RAM 4 Go, L2 4 Mio per thread. Parallelization: run 8 local-search threads independently and keep the best solution.

Instances	Sequential 60 mn	Parallel 60 mn
A1	1.694 499 e11	1.694 476 e11
A2	1.459 838 e11	1.459 712 e11
A3	1.543 302 e11	1.543 302 e11
A4	1.114 908 e11	1.114 903 e11
A5	1.245 716 e11	1.245 290 e11
B6	8.400 189 e10	8.391 967 e10
B7	8.121 408 e10	8.112 826 e10
B8	8.196 477 e10	8.196 477 e10
B9	8.185 367 e10	8.185 367 e10
B10	7.805 572 e10	7.799 984 e10