A method for constrained multiobjective optimization based on SQP techniques

Jörg Fliege[1] A. Ismael F. Vaz[2]

[1]University of Southampton, UK

[2]University of Minho, Portugal

ICATT2016

March 14-17

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x \in \Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization ($p, q = 0, \ell = -\infty^n, u = \infty^n$).

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x \in \Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization $(p, q = 0, \ell = -\infty^n, u = \infty^n)$.

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x \in \Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization $(p, q = 0, \ell = -\infty^n, u = \infty^n)$.

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x \in \Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization $(p, q = 0, \ell = -\infty^n, u = \infty^n)$.

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x \in \Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization $(p, q = 0, \ell = -\infty^n, u = \infty^n)$.

Multiobjective constrained optimization

Constrained multiobjective optimization problem

$$\min_{x\in\Omega} \quad f(x) = (f_1(x), \ldots, f_m(x))^T$$

with

$$\Omega = \{x \in [\ell, u] \subseteq \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, p, \quad h_l(x) = 0, l = 1, \ldots, q\}$$

- $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$;
- Several objectives, often conflicting.
- All objective functions are at least C^2;
- All constraint functions are at least C^1;
- The approach is valid for unconstrained optimization $(p, q = 0, \ell = -\infty^n, u = \infty^n)$.

Outline

1 Introduction

2 The algorithm

3 Implementation

4 Numerical results

5 Numerical results – real-applications on Space Engineering

6 Conclusions

Algorithm main lines

- Does not aggregate any of the objective functions
- Uses SQP based techniques for MOO
- Keeps a list of nondominated points
- Constraints violations are considered as additional objectives
- Tries to capture the whole Pareto front from two algorithmic stages: search and refining

Algorithm main lines

- Does not aggregate any of the objective functions
- Uses SQP based techniques for MOO
- Keeps a list of nondominated points
- Constraints violations are considered as additional objectives
- Tries to capture the whole Pareto front from two algorithmic stages: search and refining

Algorithm main lines

- Does not aggregate any of the objective functions
- Uses SQP based techniques for MOO
- Keeps a list of nondominated points
- Constraints violations are considered as additional objectives
- Tries to capture the whole Pareto front from two algorithmic stages: search and refining

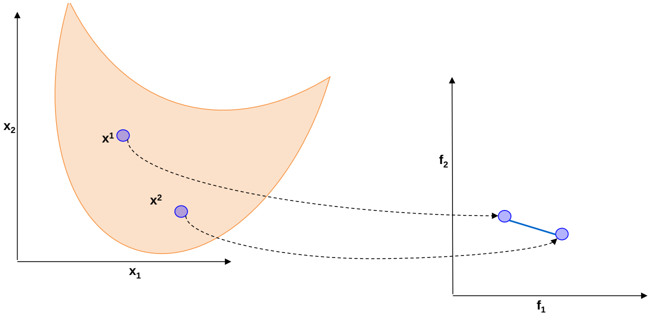
Algorithm main lines

- Does not aggregate any of the objective functions
- Uses SQP based techniques for MOO
- Keeps a list of nondominated points
- Constraints violations are considered as additional objectives
- Tries to capture the whole Pareto front from two algorithmic stages: search and refining

Algorithm main lines

- Does not aggregate any of the objective functions
- Uses SQP based techniques for MOO
- Keeps a list of nondominated points
- Constraints violations are considered as additional objectives
- Tries to capture the whole Pareto front from two algorithmic stages: search and refining

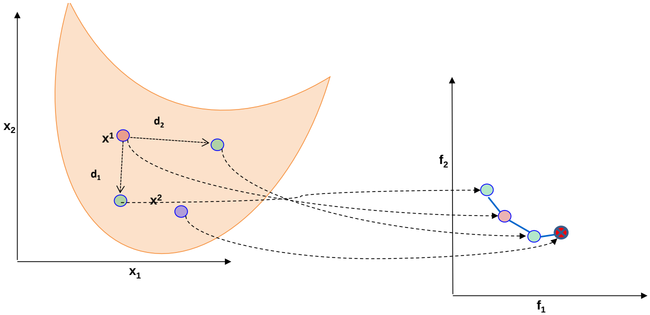
Algorithm illustrated - setup



A two dimensional ($n = 2$ and $m = 2$) example.

List of points at a given iteration $\{x^1, x^2\}$.

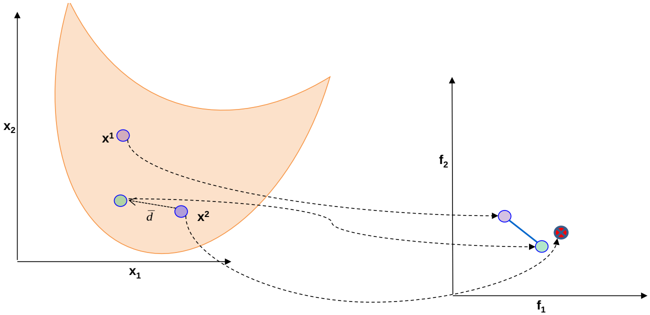
Algorithm illustrated - spread



d_i is a descent direction for f_i

The new computed point $x^1 + d_i$ (in fact $x^1 + \alpha d_i$) will not be dominated by x^1 (but may dominate or be dominated by other points in the list)

Algorithm illustrated - refining



$\bar{d}$ is a descent (non-ascending) direction for all objective functions

The new computed point $x^2 + \bar{d}$ will dominate x^2 (and may dominate or be dominated by other points in the list)

Search direction computation

For each x_k in the list of nondominated points:

Spread $(i = 1, \ldots, m)$

$$\begin{aligned} d_i \in arg \min_{d \in \mathbb{R}^n} \quad & \nabla f_i(x_k)^T d + \frac{1}{2} d^T H_i d \\ \text{s.t.} \quad & g_j(x_k) + \nabla g_j(x_k)^T d \leq 0, \quad j = 1, \ldots, p \\ & h_l(x_k) + \nabla h_l(x_k)^T d = 0, \quad l = 1, \ldots, q \\ & \ell \leq x_k + d \leq u \end{aligned}$$

where H_i is a positive definite matrix.

d_i is a descent direction for f_i and

$x_k + \alpha d_i$ will be a trial point for our list of nondominated points.

Search direction computation

For each x_k in the list of nondominated points:

Refining

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{m} f_i(x) \\
\text{s.t.} \quad & f_i(x) \leq f_i(x_k), \quad i = 1, \ldots, m \\
& g_j(x) \leq 0, \quad j = 1, \ldots, p \\
& h_l(x) = 0, \quad l = 1, \ldots, q
\end{aligned}
$$

Iterations of an SQP-type method for this problem are carried out, using x_k as a starting point.

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Some theoretical considerations

- From spread stage we are obtaining new (nondominated) points
- The spread stage performs a finite number of iterations (we are not looking for a too big – unpractical – Pareto front approximation)
- The refining stage drives all the available list points to Pareto criticality,
- by obtaining a new point that improves (decreases or maintains) all the objective function values
- (Local) Pareto criticality is possible to verify based on the refining single-objective optimization problem
- A convergence theory is available for the proposed algorithm

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_k)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_k)$ in the refining stage
- Two (list) initialization strategies are implemented:
  - `line` – a line between ℓ and u ([illegible])
  - `rand` – a uniform (ℓ, u) random distribution

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
- Two (list) initialization strategies are implemented:

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
- Two (list) initialization strategies are implemented:

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:
  - `line` – a line between ℓ and u ($x_i = \ell + i\frac{u-\ell}{2nS}$, $i = 1, \ldots, 2nS$);
  - `rand` – a uniform (ℓ, u) random distribution

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:
  - `line` – a line between ℓ and u ($x_i = \ell + i\frac{u-\ell}{2n_S}$, $i = 1, \ldots, 2n\mathcal{S}$).
  - `rand` – a uniform (ℓ, u) random distribution.

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:
  - `line` – a line between ℓ and u ($x_i = \ell + i\frac{u-\ell}{2n_{\mathcal{S}}}$, $i = 1, \ldots, 2n\mathcal{S}$).
  - `rand` – a uniform (ℓ, u) random distribution.

Implementation

- Implemented in MATLAB (fast coding, high computational time)
- (Single-objective) Subproblems are solved by `quadprog` and `fmincon` MATLAB solvers
- Maximum of 20 iterations on the spread stage
- We consider three possibilities for the H_i matrix:
  - $H_i = I_m$ in both stages
  - $H_i = (\nabla^2 f_i(x_k) + E_i)$ in both stages
  - $H_i = I_m$ in the spread stage and $H_i = (\nabla^2 f_i(x_k) + E_i)$ in the refining stage
- Two (list) initialization strategies are implemented:
  - `line` – a line between ℓ and u ($x_i = \ell + i\frac{u-\ell}{2n_{\mathcal{S}}}$, $i = 1, \ldots, 2n\mathcal{S}$).
  - `rand` – a uniform (ℓ, u) random distribution.

Outline

1. Introduction
2. The algorithm
3. Implementation
4. **Numerical results**
5. Numerical results – real-applications on Space Engineering
6. Conclusions

Test set

- In fact the majority of the MOO test problems used in the literature are suitable for our approach (derivatives are available).
- Some problems were not differential at one point of the feasible region and we consider an adapted problem (i.e. $\sqrt{x}$ at $x = 0$ and we consider $\ell = 0.001$).
- The problems are coded in AMPL (and a MATLAB-AMPL interface was used). Exact derivatives are provided by AMPL.
- 67 problems (50 problems with $m = 2$, 17 problems with $m = 3$), n varying between 2 and 30.
- 21 test problems (12 problems with $m = 2$, 9 problems with $m = 3$), 7 with nonlinear constraints, 9 with linear constraints, and 5 with both, n varying between 2 and 20.

Test set

- In fact the majority of the MOO test problems used in the literature are suitable for our approach (derivatives are available).
- Some problems were not differential at one point of the feasible region and we consider an adapted problem (i.e. $\sqrt{x}$ at $x = 0$ and we consider $\ell = 0.001$).
- The problems are coded in AMPL (and a MATLAB-AMPL interface was used). Exact derivatives are provided by AMPL.
- 67 problems (50 problems with $m = 2$, 17 problems with $m = 3$), n varying between 2 and 30.
- 21 test problems (12 problems with $m = 2$, 9 problems with $m = 3$), 7 with nonlinear constraints, 9 with linear constraints, and 5 with both, n varying between 2 and 20.

Test set

- In fact the majority of the MOO test problems used in the literature are suitable for our approach (derivatives are available).
- Some problems were not differential at one point of the feasible region and we consider an adapted problem (i.e. $\sqrt{x}$ at $x = 0$ and we consider $\ell = 0.001$).
- The problems are coded in AMPL (and a MATLAB-AMPL interface was used). Exact derivatives are provided by AMPL.
- 67 problems (50 problems with $m = 2$, 17 problems with $m = 3$), n varying between 2 and 30.
- 21 test problems (12 problems with $m = 2$, 9 problems with $m = 3$), 7 with nonlinear constraints, 9 with linear constraints, and 5 with both, n varying between 2 and 20.

Test set

- In fact the majority of the MOO test problems used in the literature are suitable for our approach (derivatives are available).
- Some problems were not differential at one point of the feasible region and we consider an adapted problem (i.e. $\sqrt{x}$ at $x = 0$ and we consider $\ell = 0.001$).
- The problems are coded in AMPL (and a MATLAB-AMPL interface was used). Exact derivatives are provided by AMPL.
- 67 problems (50 problems with $m = 2$, 17 problems with $m = 3$), n varying between 2 and 30.
- 21 test problems (12 problems with $m = 2$, 9 problems with $m = 3$), 7 with nonlinear constraints, 9 with linear constraints, and 5 with both, n varying between 2 and 20.

Test set

- In fact the majority of the MOO test problems used in the literature are suitable for our approach (derivatives are available).
- Some problems were not differential at one point of the feasible region and we consider an adapted problem (i.e. $\sqrt{x}$ at $x = 0$ and we consider $\ell = 0.001$).
- The problems are coded in AMPL (and a MATLAB-AMPL interface was used). Exact derivatives are provided by AMPL.
- 67 problems (50 problems with $m = 2$, 17 problems with $m = 3$), n varying between 2 and 30.
- 21 test problems (12 problems with $m = 2$, 9 problems with $m = 3$), 7 with nonlinear constraints, 9 with linear constraints, and 5 with both, n varying between 2 and 20.

Solvers

- We consider six implementations of the MOSQP solver: MOSQP ($H = I$, `line`), MOSQP ($H = \nabla^2 f$, `line`), MOSQP ($H = (I, \nabla^2 f)$, `line`), MOSQP ($H = I$, `rand`), MOSQP ($H = \nabla^2 f$, `rand`), MOSQP ($H = (I, \nabla^2 f)$, `rand`).
- The MOSQP solver is compared against NSGA-II (C version) and MOScalar.
- We report extensive numerical results using performance and data profiles.

Solvers

- We consider six implementations of the MOSQP solver: MOSQP ($H = I$, line), MOSQP ($H = \nabla^2 f$, line), MOSQP ($H = (I, \nabla^2 f)$, line), MOSQP ($H = I$, rand), MOSQP ($H = \nabla^2 f$, rand), MOSQP ($H = (I, \nabla^2 f)$, rand).
- The MOSQP solver is compared against NSGA-II (C version) and MOScalar.
- We report extensive numerical results using performance and data profiles.
  - For performance profiles we consider the *Purity*, *Spread* – Gamma Metric, *Spread* – Delta Metric, and the *Hypervolume* metrics.
  - While data profile indicate how likely is an algorithm to solve a problem, given some computational budget.

Solvers

- We consider six implementations of the MOSQP solver: MOSQP ($H = I$, line), MOSQP ($H = \nabla^2 f$, line), MOSQP ($H = (I, \nabla^2 f)$, line), MOSQP ($H = I$, rand), MOSQP ($H = \nabla^2 f$, rand), MOSQP ($H = (I, \nabla^2 f)$, rand).
- The MOSQP solver is compared against NSGA-II (C version) and MOScalar.
- We report extensive numerical results using performance and data profiles.
  - For performance profiles we consider the *Purity*, *Spread* – Gamma Metric, *Spread* – Delta Metric, and the *Hypervolume* metrics.
  - While data profile indicate how likely is an algorithm to ‘solve’ a problem, given some computational budget.

Solvers

- We consider six implementations of the `MOSQP` solver: `MOSQP` $(H = I,$ `line`), `MOSQP` $(H = \nabla^2 f,$ `line`), `MOSQP` $(H = (I, \nabla^2 f),$ `line`), `MOSQP` $(H = I,$ `rand`), `MOSQP` $(H = \nabla^2 f,$ `rand`), `MOSQP` $(H = (I, \nabla^2 f),$ `rand`).

- The `MOSQP` solver is compared against `NSGA-II` (C version) and `MOScalar`.

- We report extensive numerical results using performance and data profiles.

  - For performance profiles we consider the *Purity*, *Spread* – Gamma Metric, *Spread* – Delta Metric, and the *Hypervolume* metrics.

  - While data profile indicate how likely is an algorithm to 'solve' a problem, given some computational budget.

Solvers

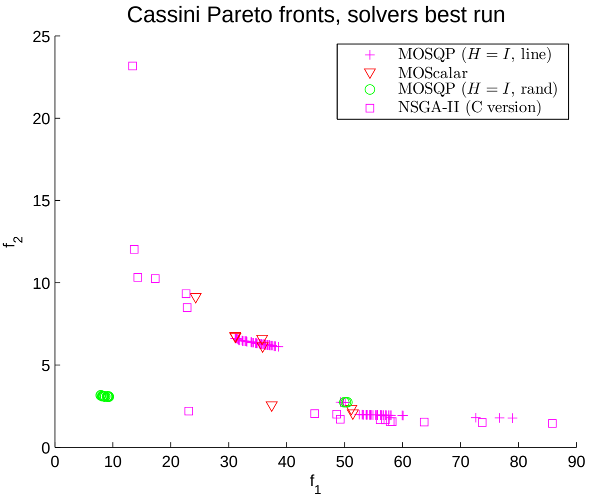
- We consider six implementations of the MOSQP solver: MOSQP $(H = I,$ line$)$, MOSQP $(H = \nabla^2 f,$ line$)$, MOSQP $(H = (I, \nabla^2 f),$ line$)$, MOSQP $(H = I,$ rand$)$, MOSQP $(H = \nabla^2 f,$ rand$)$, MOSQP $(H = (I, \nabla^2 f),$ rand$)$.
- The MOSQP solver is compared against NSGA-II (C version) and MOScalar.
- We report extensive numerical results using performance and data profiles.
  - For performance profiles we consider the *Purity*, *Spread* – Gamma Metric, *Spread* – Delta Metric, and the *Hypervolume* metrics.
  - While data profile indicate how likely is an algorithm to 'solve' a problem, given some computational budget.

Outline

1. Introduction
2. The algorithm
3. Implementation
4. Numerical results
5. Numerical results – real-applications on Space Engineering
6. Conclusions

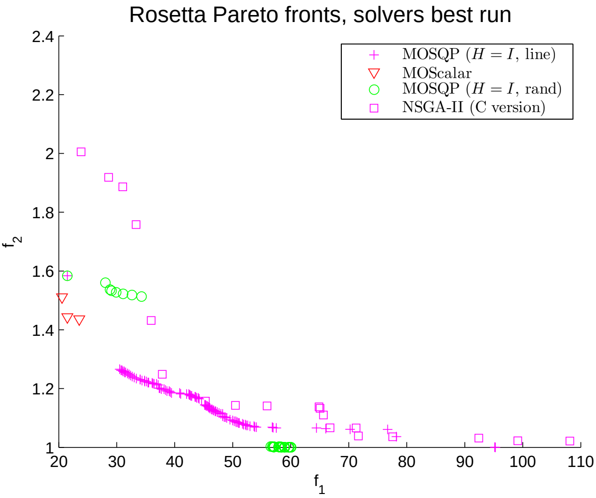
Cassini 1 bi-objective problem

f_1 is the total ΔV and f_2 is the squared total travel time.



Rosetta bi-objective problem

f_1 is the total ΔV and f_2 is the squared total travel time.



Outline

1 Introduction

2 The algorithm

3 Implementation

4 Numerical results

5 Numerical results – real-applications on Space Engineering

6 Conclusions

Conclusions

- Proposal of a method for constrained multi-objective optimization based on SQP (MOSQP)
- A convergence proof to (local) Pareto points is established
- Implementation of the proposed algorithm in MATLAB
- Numerical results confirm the solver competitiveness and robustness

Conclusions

- Proposal of a method for constrained multi-objective optimization based on SQP (MOSQP)
- A convergence proof to (local) Pareto points is established
- Implementation of the proposed algorithm in MATLAB
- Numerical results confirm the solver competitiveness and robustness

Conclusions

- Proposal of a method for constrained multi-objective optimization based on SQP (MOSQP)
- A convergence proof to (local) Pareto points is established
- Implementation of the proposed algorithm in MATLAB
- Numerical results confirm the solver competitiveness and robustness

Conclusions

- Proposal of a method for constrained multi-objective optimization based on SQP (MOSQP)
- A convergence proof to (local) Pareto points is established
- Implementation of the proposed algorithm in MATLAB
- Numerical results confirm the solver competitiveness and robustness

Thanks – Support

Thanks!

This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: PEst-OE/EEI/UI0319/2014.