UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Matematica

Numerical methods for optimization problems: an application to energetic districts

Metodi numerici per problemi di ottimizzazione: una applicazione ai distretti energetici

Relatore:
Prof. Stefania Bellavia

Candidato:
Elisa Riccietti

Correlatore:
Dott. Stefano Sello

Anno Accademico 2013/2014

Contents

Introduction

This thesis was carried out in collaboration with the research center "ENEL Ingegneria e Ricerca" in Pisa, Italy, and deals with the numerical solution of optimization problems arising in energy districts. An energy district is a complex compound of several machines that can produce, absorb or store electricity or heat and that is interlaced bidirectionally with the electrical grid, so that if energy produced within the district is more than that needed by loads it can be sold to the grid, or, if the quantity produced within the district is not sufficient, it can be bought from the grid. The aim of the optimization process is to find the asset of the devices providing the best management of local resources, in terms of costs. The recent news in the energy market, such as the spread of the non predictable renewable sources inside electric system, as well as the evident trend towards the involvement of distributed generation, cause management problems and affect the way the electricity system has to be safely managed. To deal with this problem, ENEL has developed the ECOS package (Enel Customer Optimization Service), and as part of it ECOS ES (ECOS Expert System) that by means of embedded algorithms, builds a model of the energy district and provides an optimized management of local resources. The aim is to provide the optimal dispatching of local energy resources on day ahead in order to minimize the cost of energy at customer site. ENEL equipped ECOS ES with a Sequential Linear Programming (SLP) solver that is shown to converge only to local minima. For this reason ENEL decided to investigate on methods that guarantee convergence to a global minimum of the problem. Among different methods available in literature Particle Swarm Optimization (PSO) methods were chosen. Then, in this thesis we have analyzed such methods, identified the version more suitable for the specific problem and implemented it in Matlab, providing the new solver as an alternative to the already implemented SLP. Then, since the SLP method implemented in ECOS was not theoretically supported, we devised a modification of it exploiting the penalty function approach. A theoretical analysis of the behaviour of this latter method has been carried out. We proved that the sequence generated approaches a point satisfying the first order optimality condition. Moreover the proposed method provides an estimation of Lagrange multipliers and this enables us to employ a criticality measure and a reliable stopping criterion. An extensive numerical experimentation has been carried out. Developed algorithms have been tested both on a benchmark problem usually used to test the performance of genetic algorithms and on optimization problems modelling synthetic energy districts. Finally the proposed approaches have been applied on a real energy district provided by Enel.

How to read this document

In Chapter 1 ECOS package is described. Namely we give motivations for developing such a package, we show how ECOS ES manages the optimization process, which are the variables being optimized and the Excel user interface by which the user can model its own energy district, entering the district data necessary for the optimization process. Various devices that can be chosen to be part of the district are also described. Section 1.2 is devoted to the description of the arising nonlinear programming problem. In the remaining sections the SLP algorithm devel-

oped by ENEL is outlined and details on how the user can collect the results of the optimization process from the Excel user interface are given.

In Chapter 2 some theoretical issues needed for the understanding of the subsequent chapters are introduced. In Section 2.1 a mathematical characterization of the solutions of a constrained optimization problem is described. Namely the Lagrangian function and KKT conditions are introduced. In Section 2.2 some basic concepts of the penalty function theory are given and some theoretical results are quoted.

In the subsequent chapters our original contribution is reported.

In Chapter 3 the SLP algorithm designed and implemented, which is based on a trust region approach and on penalty function theory, is introduced and studied from a theoretical point of view. Some implementation issues are also reported, namely the stopping criterion chosen for the algorithm and the updating strategy for the penalty parameter.

In Chapter 4 PSO methods are introduced and described. A stability analysis of the dynamical system constituted by particles of the swarm is performed, our implementation of PSO method in ECOS is presented and a new variation of PSO algorithm is also proposed.

In Chapter 5 the results of the numerical tests performed on different test cases with the three developed algorithms are reported.

In Chapter 6 we report on the results of the optimization of a real industrial district in Pisa, Italy. For this specific test case, we have also performed a robustness analysis, carried out varying some input parameters, namely the sell price of energy and the thermal load.

Chapter 1

Enel Customer Optimization Service (ECOS) package

In this Chapter we describe the reasons for which the research centre "Enel Ingegneria e Ricerca" in Pisa decided to develop the ECOS package, with the aim of managing the problem of the optimization of energy districts, and its functionalities. Namely, we describe, [17], [28]:

- how the user can model its own district,
- how the problem has been formalized in ECOS,
- the arising nonlinear programming problem,
- the algorithm that is in charge of its solution,
- how the results of the optimization process are managed.

The electric system has been experiencing a dramatic evolution in the last years. Many things have changed from the past: 2020 European targets require massive renewable energy source use, energy efficiency programs implementation and energy diversification; the high penetration of distributed, non predictable renewable sources inside electric system as well as the evident trend towards the involvement of distributed generation are already causing management problems, affecting the way the electricity system has to be safely managed.

On top of that, the electricity market liberalization and energy efficiency policies, coupled with macro economic situation in Europe, are fostering the end-users, either residential, commercial or industrial, to increase their attention towards a smarter approach to energy consumption. Choosing among different retailers and energy tariff schemes, planning energy demand patterns according to the actual prices, evaluating process efficiency, are part of future consumers behaviours.

According to this approach, a new smarter electricity system management is required, evolving from a paradigm in which energy was generated when required by the loads, to an optimized one in which also loads that offer a customizable logic can follows energy generation fluctuations, [17].

In order to support such an evolution, new enabling technologies are needed. According to this scenario, the research centre "Enel Ingegneria e Ricerca" in Pisa decided to develop the **ECOS package**, aimed at the optimized management of energy resources for *energy districts* and smart communication functionalities. With the term *energy district* we address any complex in which there are devices that may produce, store or absorb energy (electricity and/or heat), and that could be interlaced bidirectionally with an electrical grid, so that any electrical power generation excess can be sold to grid and likewise the district can buy electrical power from the grid, if the amount generated within it is less than that required by the users. It is also assumed

that some of the machines in the district offer a customizable control logic. Also a house could be considered an energy district, but of course this package is mainly intended for business customers, because they are characterize by significant installed capacity, flexibility margins and higher controllable resources in loads and generation capabilities, and are the one expected to gain more earnings from a smarter management of their resources, [28].

As part of the ECOS package, the **ECOS Expert System** (ECOS ES) has been developed. ECOS ES takes as an input:

- the expected load/generation profile of the district,
- the price profile for energy purchase and sell (either fixed by retailer or following the day ahead market prices),
- the economic incentives to electric energy production by renewable sources,
- the weather forecast that affect the renewable energy,
- the capabilities and constraints of the customer.

Then, ECOS ES, by means of embedded algorithms, builds a model of the energy district and provides an optimized management of local resources, providing the optimal dispatching of local energy resources on day ahead, in order to minimize the cost of energy at customer site.

The optimization tool, indeed, generates a plan, for the subsequent day, of the *set points* of each device in the district in a way that a number of requirements are fulfilled and the overall cost of energy is minimized. *Set points* are dimensionless parameters, which are directly related to the physical quantities actually under control (e.g. the electrical power produced by co-generators, that stored by batteries, the thermal power provided by boiler), and are the unknown quantity the optimization process aims to determine.

Figure 1.0.1 shows an overview of the ECOS ES input/output functional scheme. ECOS ES package has been developed both in Matlab and in C++.

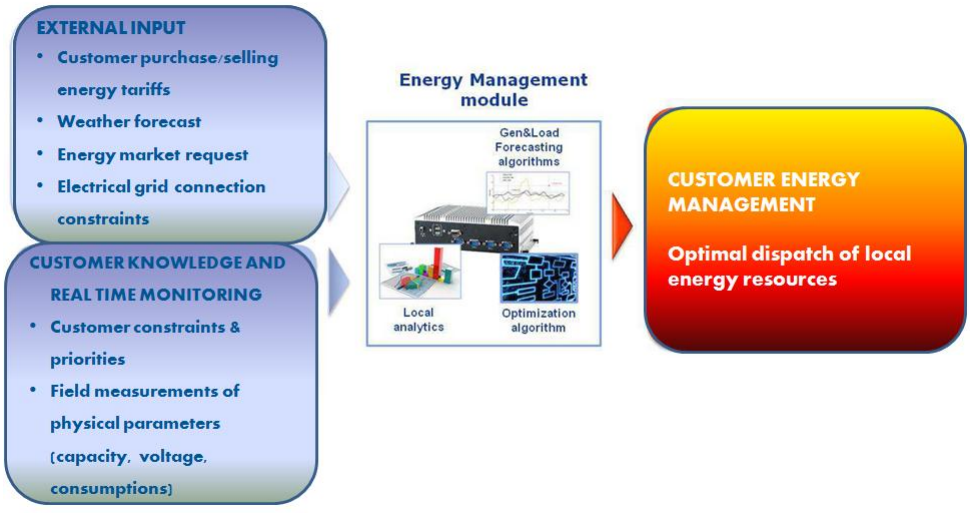


Figure 1.0.1: *ECOS ES input/output functional scheme.*

Figure (1.0.2) shows the general architecture of ECOS in its most general configuration: the Enel Server on the left, a local intelligence with ECOS ES package at the centre, and the customer assets on the right. In the drawing, the full functionalities of the ECOS concept are represented, including, for example, advices to the customer like smart actions to lower consumption and bi-directional data exchange. ECOS ES could be installed on a local intelligence,

like in Figure 1.0.2, or remotely, at retailers centralized premises, taking in charge the optimized management of all of the customers assets, according to the implemented business model. In this latter case, these informations are sent directly to the customer facilities, via an appropriate communication infrastructure that connect them to the Enel server [17].

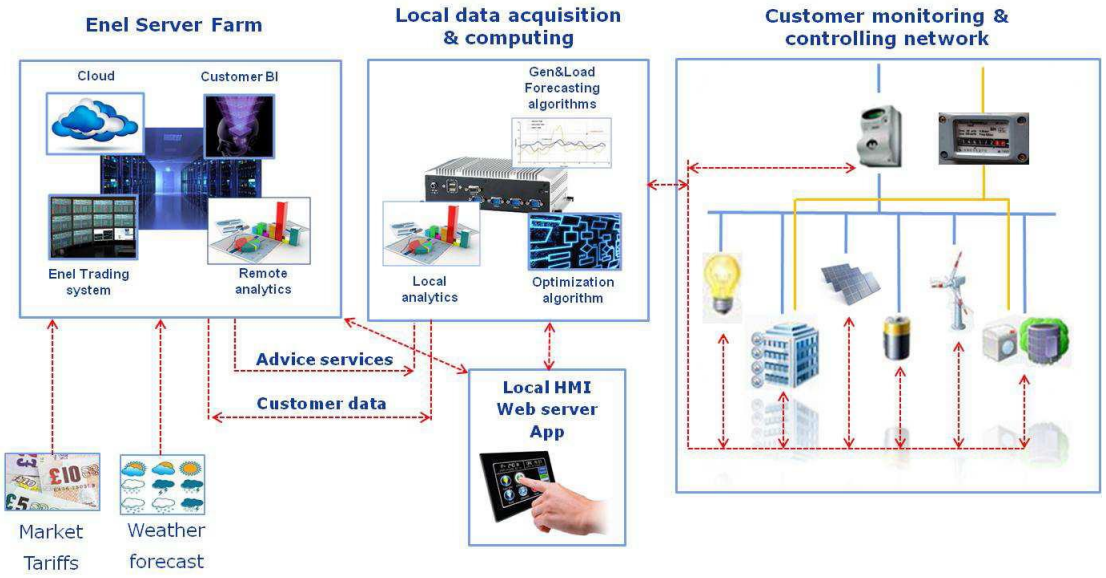


Figure 1.0.2

In the following Sections we describe in more details the functionalities of ECOS ES and the embedded optimization algorithm.

1.1 Building the district

In this Section we describe how the model of energy district is built in ECOS ES and the dedicated embedded Excel interface, [28].

Before starting the optimization process, the user has to specify the outline of the district, i.e. the devices of which it is compound and how they are connected to each other, choosing among different predefined thermal configurations for the heat/cold management. Particularly there are four different types of machines that can be included by ECOS ES as part of an energy district:

- electrical generators;
- accumulators;
- electrical loads;
- thermal configurations (hot and cold).

To do this the software tool is equipped with an Excel user interface. Each district is defined in a single Excel workbook and each workbook is composed by several sheets that must be filled by the user. In particular there is one mandatory sheet that contains all common parameters of the district environment, namely: the price of the electricity (to sell or to buy), wind speed,

solar radiation and ambient temperature. Then the user can model the district adding a sheet for each device, and it is also possible to include different machines of the same type. Those sheets are to be filled with characteristic parameters (scalar and look-up tables) of the devices to simulate their real behaviour, and also the set points of the starting district configuration must be specified. ECOS ES defines the optimal generation, load and energy storage profiles by calculating the optimal set points for each device. The optimization aims to find the best configuration for the whole next day, so if the optimization is carried on on day n it will find optimal set points for all the 24 hours of day $n + 1$. It was decided to set the time unit to be $\tau = 15$ minutes, so that the day is divided into $N = \frac{24\,60}{15} = 96$ quarters of an hour and the solver has to find the optimal set point for each device and for each time unit. As a future improvement ECOS ES will be able to work in real time, i.e. to recalculate set points, according to unexpected variations in demand/generation profiles or in weather forecast, also for a smaller time interval, for example the remaining part of the day, and it will possible to choose the time step. Also for this reason it is important to have an accurate and fast optimization method.

Below we describe in more details every kind of device that can be part of the energy district, the set points characterizing each specific machine and the constraints that the set points have to satisfy. Let i be the time index, $i = 1 \dots N = 96$, k the index of generators and thermal configurations, $k = 1 \dots N_{gen}$, b the index of accumulators, $b = 1 \dots N_{acc}$ and m the index of loads, $m = 1 \dots N_{loads}$, where $N_{gen}, N_{acc}, N_{loads}$ are respectively the total number of generators (including hot and cold configurations), accumulators and loads in the district. We also report some examples of characteristic input parameters for each device. See [6] for a more detailed description.

Electrical generators

We here consider three different types of generators:

- photovoltaic generators;
- wind turbine generators;
- fuel burning generators.

For k-th generator we denote with $\alpha_k \in \mathbb{R}^N$ the set points vector, $k = 1 \dots, N_{gen}$. Its components $\alpha_k(i)$ are the fraction of power at i-th time unit $P_k(i)$ $i = 1, \dots, N$, with respect to the rated output PN_k, to which the generator has to work:

$$\alpha_k(i) = \frac{P_k(i)}{PN_k} \tag{1.1.1}$$

The following bound constraints have to be satisfied:

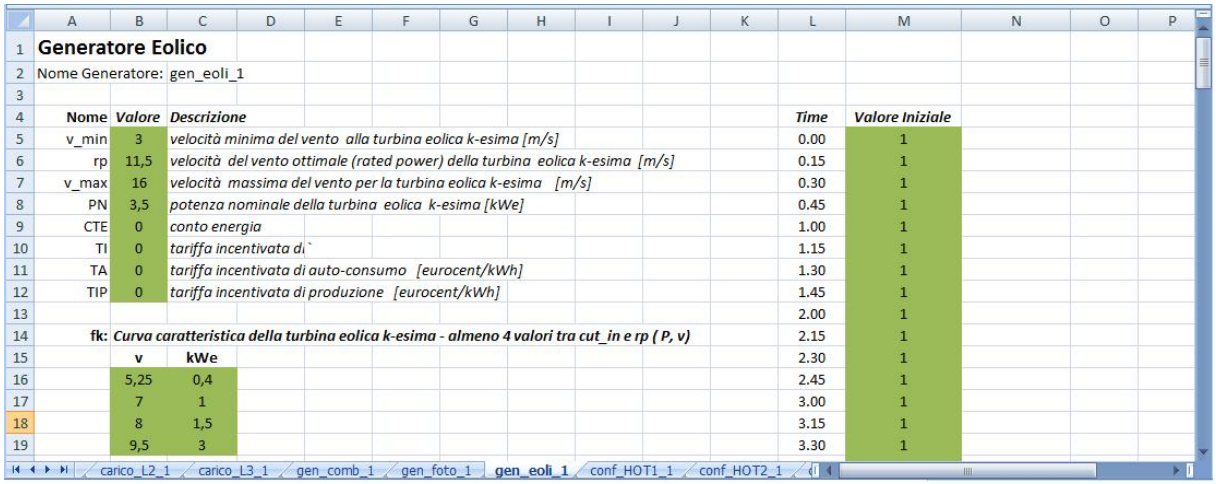
$$0 \leq \alpha_k(i) \leq 1; \tag{1.1.2}$$

where $\alpha_k(i) = 0$ means that the generator is off, $\alpha_k(i) = 1$ means that the generator works on maximum power. Note that for fuel burning generators there is also a physical constraint on the number of ignitions NI, namely NI cannot be grater than a fixed number NI_{max}, the maximum number of ignitions. Notice that NI is a nonlinear function of $\alpha_k(i)$, so this generates the following nonlinear constraint:

$$NI(\alpha_k) \leq NI_{max}. \tag{1.1.3}$$

Characteristic input parameters for this kind of devices are the rated output PN, incentives for renewable energy sources, fuel type (gas, diesel, biomass) and fuel cost and some tabulated

values of characteristic curves, for example power changing with wind velocity for wind farms or characteristic curve of the electrical efficiency for fuel burning generators. We can see in Figure 1.1.1, an example of Excel input data sheet for wind generators, in which parameters in green, i.e. scalar physical parameters, initial set points and tabled data, are to be filled to enable ECOS ES to manage the optimization process.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	**Generatore Eolico**															
2	Nome Generatore: gen_eoli_1															
3																
4	**Nome**	***Valore***	***Descrizione***									***Time***	***Valore Iniziale***			
5	v_min	3	*velocità minima del vento alla turbina eolica k-esima [m/s]*									0.00	1			
6	rp	11,5	*velocità del vento ottimale (rated power) della turbina eolica k-esima [m/s]*									0.15	1			
7	v_max	16	*velocità massima del vento per la turbina eolica k-esima [m/s]*									0.30	1			
8	PN	3,5	*potenza nominale della turbina eolica k-esima [kWe]*									0.45	1			
9	CTE	0	*conto energia*									1.00	1			
10	TI	0	*tariffa incentivata di*									1.15	1			
11	TA	0	*tariffa incentivata di auto-consumo [eurocent/kWh]*									1.30	1			
12	TIP	0	*tariffa incentivata di produzione [eurocent/kWh]*									1.45	1			
13												2.00	1			
14	**fk:** ***Curva caratteristica della turbina eolica k-esima - almeno 4 valori tra cut_in e rp (P, v)***											2.15	1			
15		**v**	**kWe**									2.30	1			
16		5,25	0,4									2.45	1			
17		7	1									3.00	1			
18		8	1,5									3.15	1			
19		9,5	3									3.30	1			

carico_L2_1 | carico_L3_1 | gen_comb_1 | gen_foto_1 | gen_eoli_1 | conf_HOT1_1 | conf_HOT2_1

Figure 1.1.1: *Input: set points and scalar parameters for a wind farm.*

Electrical accumulators

Electrical accumulators are used in the district to store electricity. Let $\beta_b \in \mathbb{R}^N$ be the set points vector for the b-th accumulator, $b = 1, \ldots, N_{acc}$. Its components $\beta_b(i)$ are the optimal set point at the i-th time unit, $i = 1, \ldots, N$. In this case set points are defined as:

$$\beta_b(i) = \frac{\sum_{k=1}^{N_{dev}} \delta(k, b, i) PN_k}{PB_b}, \tag{1.1.4}$$

where $\delta(k, b, i)$ is a function of k, b, i, PN_k is the rated output of the k-th device, N_{dev} is the total number of devices, PB_b is the rated output of the b-th battery, [6]. In this case we have the following bound constraints:

$$-1 \leq \beta_b(i) \leq 1, \tag{1.1.5}$$

where negative values of $\beta_b(i)$ means that the accumulator is providing power, positive values of $\beta_b(i)$ means that the accumulator is gaining power. For each accumulator and for each time unit there are also two physical restrictions on the state of charge $SOC_b(i)$, that is a nonlinear function of $\beta_k(i)$. Then, the following nonlinear constraint arises:

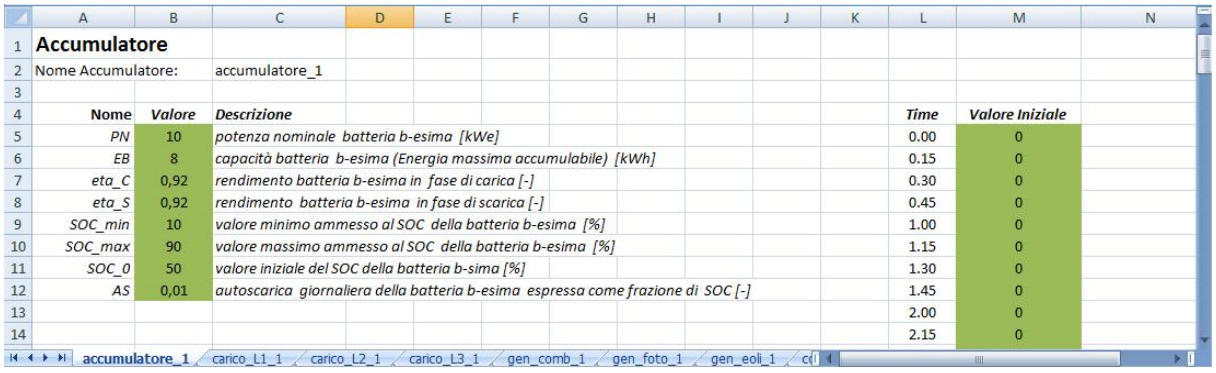
$$SOC_{bmin} \leq SOC_b(i) \leq SOC_{bmax}. \tag{1.1.6}$$

In Figure 1.1.2, an example of Excel input data sheet for accumulators.

Electrical loads

We here consider three different types of electrical loads:

- L1 loads, that are mandatory electrical consumptions;
- L2 loads, that are electrical cycles that need to be completed one or more times at no specific time in the day,



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**Accumulatore**													
2	Nome Accumulatore:		accumulatore_1											
3														
4	**Nome**	***Valore***	***Descrizione***									***Time***	***Valore Iniziale***	
5	*PN*	10	*potenza nominale batteria b-esima [kWe]*									0.00	0	
6	*EB*	8	*capacità batteria b-esima (Energia massima accumulabile) [kWh]*									0.15	0	
7	*eta_C*	0,92	*rendimento batteria b-esima in fase di carica [-]*									0.30	0	
8	*eta_S*	0,92	*rendimento batteria b-esima in fase di scarica [-]*									0.45	0	
9	*SOC_min*	10	*valore minimo ammesso al SOC della batteria b-esima [%]*									1.00	0	
10	*SOC_max*	90	*valore massimo ammesso al SOC della batteria b-esima [%]*									1.15	0	
11	*SOC_0*	50	*valore iniziale del SOC della batteria b-sima [%]*									1.30	0	
12	*AS*	0,01	*autoscarica giornaliera della batteria b-esima espressa come frazione di SOC [-]*									1.45	0	
13												2.00	0	
14												2.15	0	

accumulatore_1 | carico_L1_1 | carico_L2_1 | carico_L3_1 | gen_comb_1 | gen_foto_1 | gen_eoli_1

Figure 1.1.2: *Input: initial set point and scalar parameters for an electrical accumulator.*

- L3 loads, that are normally on, and can be shut down for a limited amount of time without compromising the related process operation.

L1 loads are given parameters for each time unit, so these loads do not have set points associated and are not included in the count of total number of loads that is $N_{loads} = N_{L2} + N_{L3}$ where N_{L2} and N_{L3} are total numbers of L2 and L3 loads respectively.

We denote with γ_m the vector of set points for m-th (L2 or L3) load. In this case set points are not time dependent. For L2 loads, indeed, set points are scalar variables in $(0, 1]$ related to the starting times of cycles i_l, so that $\gamma_m \in \mathbb{R}^{N_m}$, where N_m is the number of cycles that need to be completed by m-th L2 load. On L2 loads set points we have the following bound constraints:

$$\frac{1}{N} \leq \gamma_m(l) \leq 1, \qquad l = 1, \ldots, N_m, \tag{1.1.7}$$

and those set points are then used to compute the starting time of l-th cycle:

$$i_l = \lceil \gamma_m(l) N \rceil, \qquad l = 1, \ldots, N_m, \tag{1.1.8}$$

so that $i_l \in \{1, 2, \ldots, N\}$.

For m-th L3 load the vector of set points $\gamma_m \in \mathbb{R}^{2NI_m}$, with NI_m maximum number of ignitions. On L3 loads set points we have the following bound constraints:

$$\frac{1}{N} \leq \gamma_m(j) \leq 1, \qquad j = 1, \ldots, 2NI_m. \tag{1.1.9}$$

The odd components of γ_m are related to switch-off times s_l and the even ones to switch-on times a_l, for $l = 1, 2, \ldots, NI_m$:

$$s_l = \lceil \gamma_m(2l-1) N \rceil, \tag{1.1.10}$$
$$a_l = \lceil \gamma_m(2l) N \rceil. \tag{1.1.11}$$

On loads set points there are also some physical nonlinear constraints controlling that, for L2 loads:

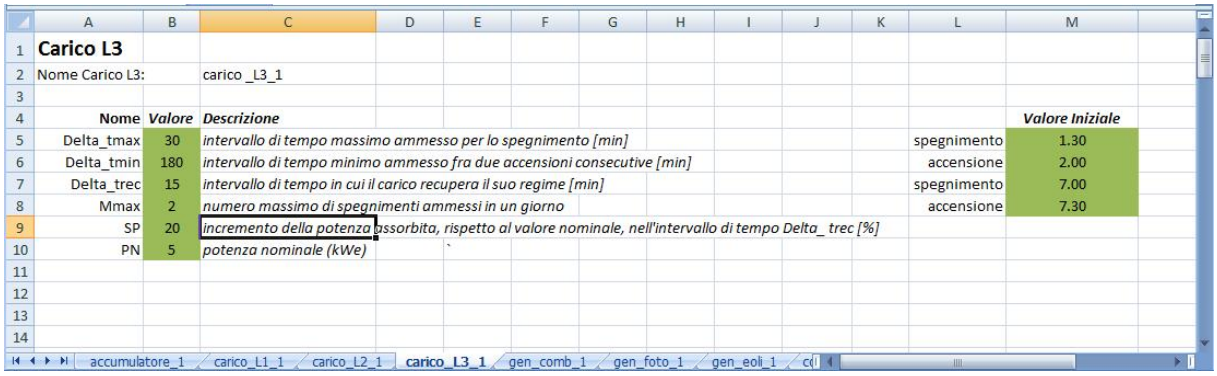
- the time between the starting points of two successive cycles is enough to complete the first of them;
- the time between the beginning of the last cycle and the end of the day is enough to complete the cycle;

for L3 loads:

- the shut down of the load precedes the turn on;

- the load is not off for more than a given amount of time;
- if a load is shut down and then turned on, a certain amount of time passes until it is turned down again.

Input parameters in this case are, for example, the number of cycles for L2 loads, maximum number of daily switching off and rated output for L3 loads. In Figure 1.1.3 an example of Excel input data sheet for L3 loads.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Carico L3**												
2	Nome Carico L3:		carico _L3_1										
3													
4	**Nome**	***Valore***	***Descrizione***										***Valore Iniziale***
5	Delta_tmax	30	*intervallo di tempo massimo ammesso per lo spegnimento [min]*									spegnimento	1.30
6	Delta_tmin	180	*intervallo di tempo minimo ammesso fra due accensioni consecutive [min]*									accensione	2.00
7	Delta_trec	15	*intervallo di tempo in cui il carico recupera il suo regime [min]*									spegnimento	7.00
8	Mmax	2	*numero massimo di spegnimenti ammessi in un giorno*									accensione	7.30
9	SP	20	*incremento della potenza assorbita, rispetto al valore nominale, nell'intervallo di tempo Delta_ trec [%]*										
10	PN	5	*potenza nominale (kWe)*										
11													
12													
13													
14													

accumulatore_1 | carico_L1_1 | carico_L2_1 | carico_L3_1 | gen_comb_1 | gen_foto_1 | gen_eoli_1

Figure 1.1.3: *Input: set points and scalar parameters for a L3 load.*

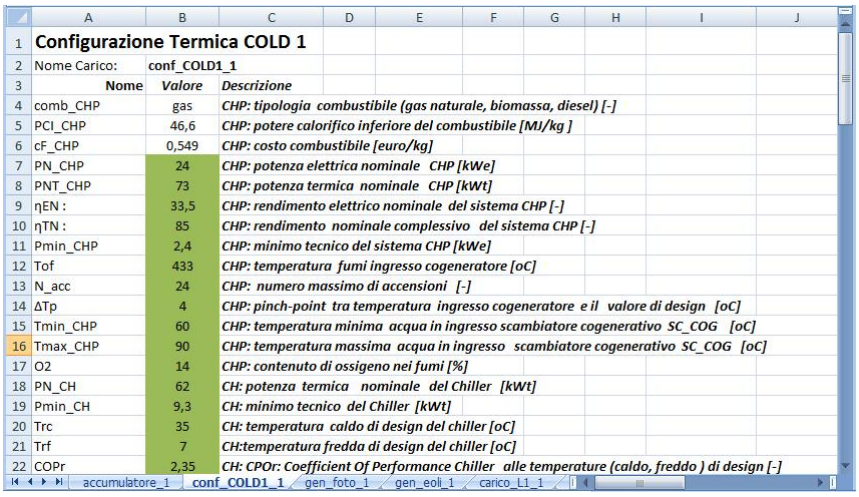
Thermal configurations

Thermal configurations are predefined configurations that can be found in an energy district for the heat/cold management, and comprise a number of thermal devices that may generate, store or absorb heat and are connected to each other in many different ways. An hot configuration is a typical winter configuration, examples of devices that might be present in an hot configuration are: CHPs (Combined Heat and Power), boilers, heat pumps, thermal accumulators. A cold configuration is a typical summer configuration, that might include: CHPs, refrigerating machines, chillers. As for generators, we denote with $\alpha_k(i)$ the set point of the k-th configuration and at the i-th time unit. The bound constraints are the same of generators, and also some of these configurations have a process constraint on the maximum number of ignitions. We notice that in any hot or cold configuration in which both a boiler and a CHP are present, in ECOS ES only CHP's set points are optimized and those of the boiler are calculated accordingly. It was decided to do this in order to restrict the number of variables to be optimized, but as a future improvement every device in the configuration will have its own set points. A wider description of each configuration available is postponed in Chapter 5.2.

In Figure 1.1.4 and Figure 1.1.5 an example of Excel input data sheet for a cold configuration is shown.

1.2 The optimization problem

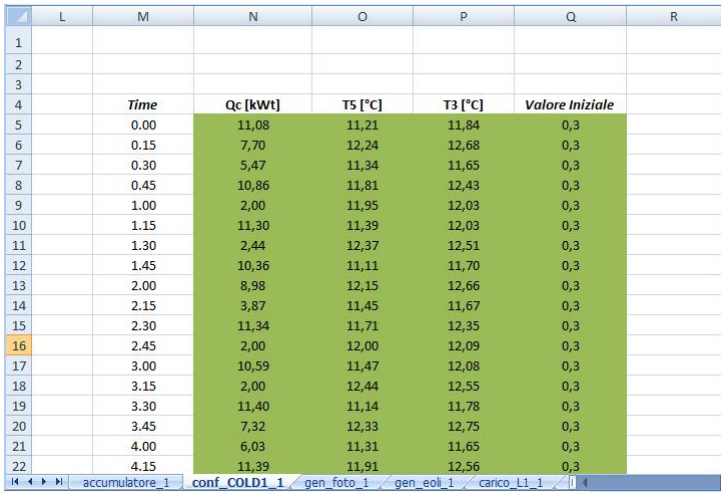
In this section we describe how the optimization of energy district is formalized in ECOS ES as a constrained nonlinear programming problem and we outline some of its features in order to make clear which kind of problem we have to solve and the difficulties related to it. Let us resume the notation introduced above that we are going to use in this section. We denote with: N_{gen}, N_{acc}, N_{L2}, N_{L3} ,N_{loads} the total number of, respectively, generators plus thermal configurations, accumulators, L2 loads, L3 loads, L2 plus L3 loads; N_m the number of cycles that need to be completed by the m-th L2 load, NI_m the maximum number of ignitions for the m-th L3 load; $N = \frac{24\ 60}{\tau} = 96$ is the total number of time units.



	A	B	C
1	**Configurazione Termica COLD 1**		
2	Nome Carico:	conf_COLD1_1	
3	**Nome**	***Valore***	***Descrizione***
4	comb_CHP	gas	*CHP: tipologia combustibile (gas naturale, biomassa, diesel) [-]*
5	PCI_CHP	46,6	*CHP: potere calorifico inferiore del combustibile [MJ/kg]*
6	cF_CHP	0,549	*CHP: costo combustibile [euro/kg]*
7	PN_CHP	24	*CHP: potenza elettrica nominale CHP [kWe]*
8	PNT_CHP	73	*CHP: potenza termica nominale CHP [kWt]*
9	ηEN :	33,5	*CHP: rendimento elettrico nominale del sistema CHP [-]*
10	ηTN :	85	*CHP: rendimento nominale complessivo del sistema CHP [-]*
11	Pmin_CHP	2,4	*CHP: minimo tecnico del sistema CHP [kWe]*
12	Tof	433	*CHP: temperatura fumi ingresso cogeneratore [oC]*
13	N_acc	24	*CHP: numero massimo di accensioni [-]*
14	ΔTp	4	*CHP: pinch-point tra temperatura ingresso cogeneratore e il valore di design [oC]*
15	Tmin_CHP	60	*CHP: temperatura minima acqua in ingresso scambiatore cogenerativo SC_COG [oC]*
16	Tmax_CHP	90	*CHP: temperatura massima acqua in ingresso scambiatore cogenerativo SC_COG [oC]*
17	O2	14	*CHP: contenuto di ossigeno nei fumi [%]*
18	PN_CH	62	*CH: potenza termica nominale del Chiller [kWt]*
19	Pmin_CH	9,3	*CH: minimo tecnico del Chiller [kWt]*
20	Trc	35	*CH: temperatura caldo di design del chiller [oC]*
21	Trf	7	*CH:temperatura fredda di design del chiller [oC]*
22	COPr	2,35	*CH: CPOr: Coefficient Of Performance Chiller alle temperature (caldo, freddo) di design [-]*

accumulatore_1 | conf_COLD1_1 | gen_foto_1 | gen_eoli_1 | carico_L1_1

Figure 1.1.4: *Input: scalar parameters for a cold configuration composed of a CHP and a chiller.*



	L	M	N	O	P	Q	R
1							
2							
3							
4		***Time***	**Qc [kWt]**	**T5 [°C]**	**T3 [°C]**	***Valore Iniziale***	
5		0.00	11,08	11,21	11,84	0,3	
6		0.15	7,70	12,24	12,68	0,3	
7		0.30	5,47	11,34	11,65	0,3	
8		0.45	10,86	11,81	12,43	0,3	
9		1.00	2,00	11,95	12,03	0,3	
10		1.15	11,30	11,39	12,03	0,3	
11		1.30	2,44	12,37	12,51	0,3	
12		1.45	10,36	11,11	11,70	0,3	
13		2.00	8,98	12,15	12,66	0,3	
14		2.15	3,87	11,45	11,67	0,3	
15		2.30	11,34	11,71	12,35	0,3	
16		2.45	2,00	12,00	12,09	0,3	
17		3.00	10,59	11,47	12,08	0,3	
18		3.15	2,00	12,44	12,55	0,3	
19		3.30	11,40	11,14	11,78	0,3	
20		3.45	7,32	12,33	12,75	0,3	
21		4.00	6,03	11,31	11,65	0,3	
22		4.15	11,39	11,91	12,56	0,3	

accumulatore_1 | conf_COLD1_1 | gen_foto_1 | gen_eoli_1 | carico_L1_1

Figure 1.1.5: *Input: thermal load and water temperature for a cold configuration.*

Once the user has built the district and entered all the necessary informations, the code is run. Specific (Matlab or C++) functions build the nonlinear cost and constraints functions, $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^p$, defining a constrained nonlinear optimization problem of the form:

$$\min_x \ f(x), \tag{1.2.1a}$$

subject to

$$x_{\min} \leq x \leq x_{\max}, \tag{1.2.1b}$$
$$g(x) \leq 0, \tag{1.2.1c}$$

where $x \in \mathbb{R}^n$, with

$$n = N[N_{gen} + N_{acc}] + \sum_{m=1}^{N_{L2}} N_m + \sum_{m=1}^{N_{L3}} NI_m. \tag{1.2.2}$$

The vector x denotes the stacked vector of all devices set points; $x_{\min} \in \mathbb{R}^n$ and $x_{\max} \in \mathbb{R}^n$ denote the associated bound constraints, from equations (1.1.2), (1.1.5), (1.1.7), (1.1.9), i.e.

$$x = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N_{gen}} \\ \beta_1 \\ \vdots \\ \beta_{N_{acc}} \\ \gamma_1 \\ \vdots \\ \gamma_{N_{loads}} \end{bmatrix}, \qquad x_{\min} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{-1} \\ \vdots \\ \mathbf{-1} \\ \frac{\mathbf{1}}{N} \\ \vdots \\ \frac{\mathbf{1}}{N} \end{bmatrix}, \qquad x_{\max} = \begin{bmatrix} \mathbf{1} \\ \vdots \\ \mathbf{1} \\ \mathbf{1} \\ \vdots \\ \mathbf{1} \\ \mathbf{1} \\ \vdots \\ \mathbf{1} \end{bmatrix},$$

in which:

- α_k for $k = 1, \ldots, N_{gen}$ is the vector of length N of the set points of the k-th generator or thermal configuration;
- β_b for $b = 1, \ldots, N_{acc}$ is the vector of length N of the set points of the b-th accumulator;
- γ_m for $m = 1, \ldots, N_{L2}$ is the vector of length N_m of the set points of m-th L2 load and γ_m for $m = N_{L2}+1, \ldots, N_{loads}$ is the vector of length $2NI_m$ of the set points of m-th L3 load. We recall that the set points corresponding to the L1 loads are not present as they are known parameters of the problem, but these parameters have to be taken into account because their presence affects the objective function.

The objective function f represents the overall daily cost of energy obtained as a result of the difference between purchase costs (fuel and electric energy) and incomings (incentives and sales revenues). It is calculated as the sum of the overall district cost of each time instant $i = 1, \ldots, N$:

$$f(x) = \sum_{i=1}^{N} \bar{f}_i(x)$$

where $\bar{f}_i(x)$ is the sum of all the partial cost functions of district devices for the i-th time unit:

$$\bar{f}_i(x) = \sum_{k=1}^{N_{gen}} f_{i,k}(x) + \sum_{b=1}^{N_{acc}} f_{i,b}(x) + \sum_{m=1}^{N_{loads}} f_{i,m}(x).$$

As discussed in Section 1.1, a number of devices have physical constraints in addition to bound constraints on their set points (see equations (1.1.3), (1.1.6)). Let us define:

- $g_k(\alpha_k)$ the constraint function of the k-th generator or thermal configuration, $g_k : \mathbb{R}^N \to \mathbb{R}$ (notice that not all generators have process constraints on their set points so $g_k(\alpha_k)$ could be also empty);
- $g_b(\beta_b)$ the constraint function for the b-th battery, $g_b : \mathbb{R}^N \to \mathbb{R}^{2N}$;
- $g_m(\gamma_m)$ the constraint function for the m-th electrical load, $g_m : \mathbb{R}^{N_m} \to \mathbb{R}^{N_m}$, for L2 loads, $g_m : \mathbb{R}^{2NI_m} \to \mathbb{R}^{3NI_m}$ for L3 loads.

These process constraint vectors can be stacked together obtaining a single vector of constraints $g : \mathbb{R}^n \to \mathbb{R}^p$, with $p = N_c + 2NN_{acc} + \sum_{m=1}^{N_{L2}} N_m + \sum_{m=1}^{N_{L3}} 3NI_m$, where N_c is the number of generator plus thermal configuration that have a process constraint on the number of ignitions:

$$g(x) = \begin{bmatrix} g_1(x) \\ \vdots \\ g_{N_{gen}}(x) \\ g_{N_{gen}+1}(x) \\ \vdots \\ g_{N_{gen}+N_{acc}}(x) \\ g_{N_{gen}+N_{acc}+1}(x) \\ \vdots \\ g_{N_{gen}+N_{acc}+N_{loads}}(x) \end{bmatrix}.$$

We here report some general characteristics of problem (1.2.1) and some implementation choices made by Enel Ingegneria e Ricerca to make problem (1.2.1) easier to solve, [28]:

- problem (1.2.1) is a nonlinear programming problem (NLP) because both the objective function f and the constraints function g are nonlinear functions;
- convexity of functions f and g does not hold in general. Moreover, the problem size is typically of the order of (at least) several hundred variables (n) and constraints ($c + 2n$);
- variables corresponding to L2 and L3 electrical loads, i.e. components of vectors γ_m for $m = 1, \ldots, N_{loads}$, should be restricted to the discrete set of values $\left\{\frac{1}{N}, \frac{2}{N}, \ldots, 1\right\}$, because we work with discrete time intervals. This would make (1.2.1) a mixed-integer nonlinear programming problem (MINLP), and it is well known that large-scale MINLP are rather difficult to solve. In order to work with continues variables in ECOS ES values of $\gamma_m(\cdot) \in \left[\frac{1}{N}, 1\right]$ are rounded internally to the nearest value of corresponding discrete set in the device calculations, so that all decision variables becomes continuous;
- functions f_k for generators and thermal configurations feature discontinuities at the minimum set point values,

$$\alpha_{\min,k} = \frac{P_{\min,k}}{PN_k}, \qquad k = 1, \ldots, N_{gen}, \tag{1.2.3}$$

  where $P_{\min,k}$ is the minimum value of power for the k-th generator, because these objective functions depends on the power produced that in turn depends on the devices status, which is a inherently discontinuous function, being defined as :

$$\theta(\alpha_k(i)) = \begin{cases} 1 & \text{if } \alpha_k(i) \geq \alpha_{\min,k} \\ 0 & \text{if } \alpha_k(i) < \alpha_{\min,k} \end{cases}. \tag{1.2.4}$$

The same problem affects the constraints functions g_k of fuel burning generators and thermal configurations with CHP, because they have a constraint on the maximum number of ignitions that involves variation of the state function $\theta(\alpha_k)$. Being these non-smooth functions, in the computation of the gradients ∇f and ∇g, in ECOS ES the state function is replaced by a continuous and differentiable sigmoid function:

$$\bar{\theta}(\alpha_k) = \frac{1}{1 + e^{-a(\alpha_k - \alpha_{k,\min})}}. \tag{1.2.5}$$

It is easy to see that $\bar{\theta}(\alpha_k)$ is monotonic increasing, and tends to $\theta(\alpha_k)$ as the parameter a increases. Based on numerical experience the value $a = 20$ has been chosen, [28]. We remark that both the objective function f and the constraint function g are still evaluated exactly using (1.2.4).

Once the optimization problem has been formalized, the embedded optimization algorithm is run in order to solve (1.2.1).

1.3 Sequential linear programming solver

In this section we describe the Sequential Linear Programming solver Enel Ingegneria e Ricerca equipped ECOS ES with, in order to solve problem (1.2.1).

Sequential Linear Programming (SLP) is an iterative method aimed at finding local optima for nonlinearly constrained optimization, that generates steps by solving successive linear subproblems, [30].

Let us here consider problem (1.2.1) and let $\Omega = \{x |\ x_{min} \leq x \leq x_{max},\ \ g(x) \leq 0\}$ be the *feasible set*. A point $x \in \Omega$ is said to be a *feasible point*.

The essential idea of SLP is to model (1.2.1) at the current iterate x_k by a linear programming subproblem and to use its minimizer to define a new iterate x_{k+1}. The challenge is to design the linear subproblem so that it yields a good step for the underlying constrained optimization problem (1.2.1) and so that the overall SLP algorithm has good convergence properties and good practical performance. So, to solve a single nonlinear optimization problem, SLP solves a sequence of linear programming problems approximating (1.2.1), by using the first order Taylor expansion of the objective function and of each nonlinear constraint function. The SLP algorithm is usually equipped with a trust region, [11], in order to confine the search to a region in which the linearised functions appear to be a good approximation of the original nonlinear ones around the current solution approximation.

Here we describe the SLP algorithm implemented in ECOS ES, [28]. At the generic iteration k, given the current approximation x_k, the first order Taylor expansion of the objective function at x_k is calculated:

$$m_k(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) \tag{1.3.1}$$

to which we will refer below as the model function and that can be rewritten as follows:

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d, \tag{1.3.2}$$

where $d = x - x_k$. Likewise nonlinear constraints functions are linearised:

$$g_{i,k}(x) = g_i(x_k) + \nabla g_i(x_k)^T (x - x_k) \qquad i = 1, \ldots, p; \tag{1.3.3}$$

or rather

$$g_{i,k}(d) = g_i(x_k) + \nabla g_i(x_k)^T d \qquad i = 1, \ldots, p. \tag{1.3.4}$$

Then, a trust region framework is used, [11]. Trust region methods define a region around the current iterate within which they trust that the linear programming problem constituted by (1.3.2) and constraint (1.3.4) is a good model for the nonlinear programming problem (1.2.1). That is, in addition to constraint (1.3.4) a trust region constraint $\|d\| \leq \Delta_k$ is considered, where Δ_k is the current trust region radius. Then, the search space is the intersection between the trust region and the feasibility region of linearised constraints. For the trust region constraint any norm can be chosen, in this case ∞-norm was used for convenience. Then, letting d_j the j-th component of vector d, the trust region constraints are linear constraints: $-\Delta_k < d_j < \Delta_k$, for $j = 1, \ldots, n$, like constraint (1.3.4), so that the problem structure is not changed. With this choice the trust region is a hypercube, [11].

The size of the trust region is critical to the effectiveness of each step. If the region is too small, the algorithm misses an opportunity to take a substantial step that will move it much closer to the minimizer of the objective function. If it is too large, the minimizer of the model in the region may be far from the minimizer of the objective function, so we may have to reduce the size of the region and try again, [30].

Then the following linear programming subproblem is solved, in order to find the optimal step defining the new solution approximation:

$$\min_{d} \; m_k(d), \tag{1.3.5a}$$

subject to

$$g_i(x_k) + \nabla g_i(x_k)^T d \leq 0; \qquad i = 1, \ldots, p, \tag{1.3.5b}$$

$$\max((x_{min} - x_k)_j, -\Delta_k) \leq d_j \leq \min((x_{max} - x_k)_j, \Delta_k), \qquad j = 1, \ldots, n. \tag{1.3.5c}$$

Note that bound constraints (1.2.1b) have been incorporated in the trust region constraint. We denote with $\bar{\Delta}_k$ a n-dimensional vector whose components are all equal to Δ_k, in order to write (1.3.5c) in a more compact form:

$$\max((x_{min} - x_k), -\bar{\Delta}_k) \leq d \leq \min((x_{max} - x_k), \bar{\Delta}_k). \tag{1.3.6}$$

Then, (1.3.5c) is replaced by (1.3.6). Let d_k be a solution of problem (1.3.5).

At each iteration it is necessary to decide whether to accept the new iterate $x_{k+1} = x_k + d_k$ or not and to choose the trust region radius for the next iteration. Both this choices are based on the agreement between the model function m_k and the objective function f. Given a step d_k, as standard in trust region approaches [11], the ratio between the *actual reduction* (numerator), and the *predicted reduction* (denominator) is defined:

$$\rho_k = \frac{f(x_k) - f(x_k + d_k)}{m_k(0) - m_k(d_k)} = \frac{f(x_k) - f(x_k + d_k)}{-\nabla f(x_k)^T d_k}. \tag{1.3.7}$$

Note that since the step d_k is obtained by minimizing the model m_k over a region that includes the step $d = 0$, the predicted reduction will always be nonnegative. Thus if ρ_k is negative, the new objective value $f(x_k + d_k)$ is greater than the current value $f(x_k)$, so the step must be rejected and $x_{k+1} = x_k$. On the other hand, if $\rho_k > \eta$, where η is a tolerance to be fixed, typically $\eta \in (0, \frac{1}{4})$ [11], the step is accepted. Regarding the trust region radius, if ρ_k is close to 1 or greater than 1, there is a good agreement between the model m_k and the function f over this step, so it is safe to expand the trust region for the next iteration. If ρ_k is positive but not close to 1, the trust region is left unchanged, while if it is close to zero or negative, the trust region is shrunk. Note that the radius is increased only if d_k actually reaches the boundary of the trust region. If the step stays strictly inside the region, we infer that the current value of Δ_k

is not interfering with the progress of the algorithm, so its value is left unchanged for the next iteration, [30].

Working with problem (1.3.5) is attractive because it requires no choice of parameters, but it has the drawback that the linearised constraints may not be consistent, that is there may be no value d for which (1.3.5b) holds, or the solution found, d_k, might be infeasible for the original nonlinear problem, i.e. $x_k + d_k \notin \Omega$. A possible solution to this, the one implemented in ECOS ES, is to reject the step if the feasible space is empty or if the provided step is not feasible, shrink the trust region and solve problem (1.3.5) again, because the smaller the region around the current solution is, the better the linearised objective function approximate the original one.

The SLP approach specifically implemented for the problem of energy district (1.2.1) is sketched in the following algorithm. The value of the tolerances used and the strategy followed to update the trust region radius are chosen according to typical heuristics [30].

Algorithm 1: SLP algorithm with trust region.

1. Set $k = 0$, specify k_{max} and x_0 such that $x_{\min} \leq x_0 \leq x_{\max}$ and $\max g(x_0) \leq 0$. Set $\epsilon = 10^{-8}$, $\epsilon_f = 10^{-2}$, $\eta = 10^{-3}$, $\rho_{bad} = 0.10$ and $\rho_{good} = 0.75$.

2. For $k = 0, \ldots, k_{max}$ perform the following steps.

   (a) Evaluate $\nabla f(x_k)$ and $\nabla g_i(x_k)$ for $i = 1, \ldots, p$. Solve the following LP:

   $$\min_d \quad f(x_k) + \nabla f(x_k)^T d \tag{1.3.8a}$$

   subject to:

   $$g_i(x_k) + \nabla g_i(x_k)^T d \leq 0 \;\; i = 1, \ldots, p, \tag{1.3.8b}$$
   $$\max((x_{min} - x_k), -\bar{\Delta}_k) \leq d \leq \min((x_{max} - x_k), \bar{\Delta}_k), \tag{1.3.8c}$$

   obtaining a candidate step d_k. If problem (1.3.8) has no solution, shrink the trust region:

   $$\Delta_{k+1} = \tfrac{1}{2}\Delta_k,$$

   and solve (1.3.8) again.

   (b) If $\|d_k\|_\infty \leq \epsilon$, stop.

   (c) Check if the step is feasible, i.e. if

   $$\max g(x_k + d_k) \leq \epsilon_f, \tag{1.3.9}$$

   holds. If (1.3.9) holds, go to step 2d. If (1.3.9) does not hold, reject the step, shrink the trust region:

   $$\Delta_{k+1} = \tfrac{1}{2}\Delta_k,$$

   and go to step 2a.

   (d) Compute the step evaluation parameter:

   $$\rho_k = \frac{f(x_k) - f(x_k + d_k)}{-\nabla f(x_k)^T d_k}. \tag{1.3.10}$$

   Then:

   i. If $\rho_k \leq \rho_{bad}$ reduce the trust-region radius:

   $$\Delta_{k+1} = \tfrac{1}{2}\Delta_k$$

   and go to step 2e.

ii. ElseIf $\rho_k \geq \rho_{good}$, and in addition $\|d_k\|_\infty \geq 0.8\Delta_k$, increase the trust-region radius

$$\Delta_{k+1} = 2\Delta_k$$

and go to step 2e.

iii. Else do not alter the trust region:

$$\Delta_{k+1} = \Delta_k$$

and go to step 2e.

(e) If $\rho_k > \eta$ accept the step d_k and calculate the new solution approximation:

$$x_{k+1} = x_k + d_k,$$

Else reject the step and set

$$x_{k+1} = x_k.$$

Go to step 2a.

In ECOS ES it was decided to adopt this strategy in order to future usage in run-time optimization. In fact, in this way the procedure can be stopped being sure that the solution is feasible and without waiting for convergence. Nevertheless this is an heuristic approach, because we are not aware of theoretical results proving the global convergence of the outlined approach. We underline also that it may happens that a linear programming problem (1.3.8) with empty feasible set is produced when reducing the trust region radius in consequence of an unsuccessful step and further reductions of the trust region radius do not help to obtain a non empty feasible set. This is particularly evident in case there are also equality constraints, and can be shown by the following example, [11].

Let $h : \mathbb{R}^n \to \mathbb{R}^m$ and

$$\theta_k = \min_{h(x_k)+J_h(x_k)^T d=0} \|d\|_2, \tag{1.3.11}$$

where $J_h(x) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix of function h. There can be no feasible point for (1.3.5) if subject also to the equality constraint $h(x_k) + J_h(x_k)^T d = 0$ whenever $\Delta_k < \theta_k$, as it is shown in Figure 1.3.1, [11].

In the left figure, the trust region radius is sufficiently large for the trust region and the linearised constraints to intersect. This is not so for the smaller trust region illustrated in the right figure. Thus, if the model gives a poor prediction of the merit function, we cannot rely on the usual mechanism of simply reducing the trust region radius to make further progress. Concerning specific case of problem (1.2.1) in which there are only inequality constraints, to help avoiding this situation it is necessary to start from a feasible point, otherwise it is likely for the LP subproblem (1.3.8) not to have a solution. Numerical tests have indeed confirmed that starting from a non feasible point the procedure may not converge. However, even starting from a feasible point, convergence of the sequence generated by Algorithm 1 sketched above is not ensured. Moreover by now it is easy to detect a feasible initial guess, i.e. a vector $x \in \Omega$, but future ECOS ES model improvement requires to add new constraints for which it may be not so simple to find a feasible starting solution.

At this stage the optimization process is stopped when the norm of the step is smaller than a fixed tolerance or when a maximum number of iterations is reached. This stopping criterion is not so satisfactory, because at the end of the optimization process we do not have any optimality measure of the provided approximate solution, in other words it is not clear at all if we have a good approximation of a solution of the original problem (1.2.1).

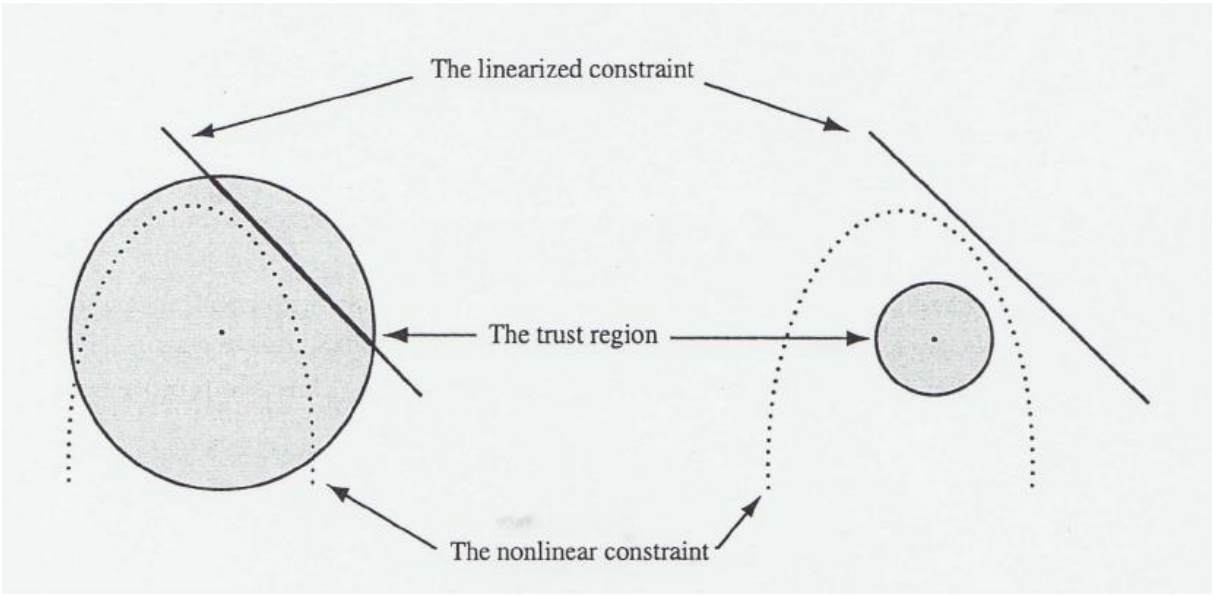


Figure 1.3.1: *The intersection between the linearization of a nonlinear constraint and a spherical trust region.*

1.4 Two phase strategy for districts that comprise batteries

In this section we describe a two phase strategy implemented in ECOS ES to overcome failures due to severe restrictions of the trust region size, [28]. In fact, based on numerical experience, when the district contains batteries, it is likely that the first SLP iterations exploit energy stored in the batteries to reduce the overall cost because no costs are associated to that energy. If the computed step induces SOC (state of charge) constraint violations, then it is rejected, the trust region size is reduced and this may prevent the solver to take good steps and the achieved point at the end of the iterative process may be too far from a good approximation of the minimum. To overcome these problems, the following strategy was chosen.

The SLP iterations are divided into two phases:

1. Phase 1: the b-th accumulator set points are fixed to the following positive value:

$$\beta_{0,b} = \frac{\frac{AS_b}{N}}{\frac{\tau}{60}\frac{PN_b}{EB_b}\eta_c} \tag{1.4.1}$$

   where AS_b is the daily auto-discharge, PN_b is the rated power, EB_b is the battery capacity, η_c is the yield charging. In this way, at each time instant the battery is slightly charged to compensate the auto-discharge and the battery SOC remains constant at all time instants $i = 1, \ldots, N$, so that SOC constraints are always satisfied. More specifically, the upper and lower bound for battery set point bound constraints $[-1, 1]$ are modified, respectively, to the values $[\beta_0(b) - 10^{-4}, \beta_0(b) + 10^{-4}]$.

2. Phase 2: the battery set point bound constraints are set to the actual values $[-1, 1]$. The algorithm is run choosing the point achieved in the first phase as a starting point.

For each phase a fixed value for the maximum number of iterations k_{max} is specified. With this two phase strategy energy stored in batteries is made available when the objective function has been already partially optimized. In this way that energy is better exploited and the accumulators model behaviour is more realistic. However as a future improvement it will be tested a one step strategy with a new term in the cost function, to be defined, associated to energy stored in batteries.

1.5 Output

In this section we describe how the results of the optimization process are managed.

After the code's run, an output Excel workbook is produced, whose sheets are filled with the results and the details of the optimization, such as optimal set points for each device, electrical power exchanged with the grid and energy costs for each time unit, and some informations about the optimization process, regarding both the decrease of the total costs and parameters specific for the solver, e.g. the trust region radius Δ_k and the ratio ρ_k at each iteration. Some examples of Excel output data sheets are shown in Figures (1.5.1), (1.5.2), (1.5.3) and (1.5.4).

	A	B	C
1	Time	Setpoint	G [kWe]
2	00:00:00	1	0
3	00:15:00	1	0
4	00:30:00	1	0
5	00:45:00	1	0
6	01:00:00	1	0
7	01:15:00	1	0
8	01:30:00	1	0
9	01:45:00	1	0
10	02:00:00	1	0
11	02:15:00	1	0
12	02:30:00	1	0
13	02:45:00	1	0
14	03:00:00	1	0

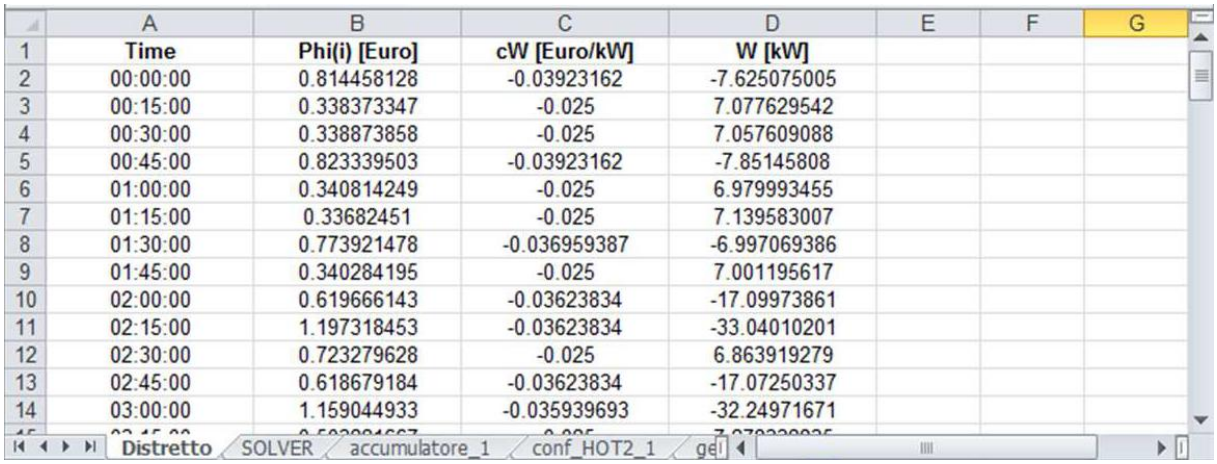
conf_HOT2_1 | gen_foto_1 | gen_eoli_1 | carico_L1_1

Figure 1.5.1: *Output: optimized set point for a photovoltaic generator*

	A	B	C	D	E	F	G	H	I
1	Time	Setpoint	G [kWe]	F_CHP [kg/h]	F_CD [kg/h]	Q_CHP [kWt]	Q_CD [kWt]	Ts [°C]	WARNING
2	00:00:00	1	24.52886926	5.226757153	0	5.226757153	0	78	WARNING: Alcuni elementi Q_CD sono minori di PMIN_CD
3	00:15:00	1	24.52886926	5.226757153	0	5.226757153	0	71.06559765	WARNING: Alcuni elementi Ts sono minori di Tmin_CHP
4	00:30:00	1	24.52886926	5.226757153	0	5.226757153	0	69.45347363	WARNING: Alcuni elementi Ts sono minori di Tmin_CD
5	00:45:00	1	24.52886926	5.226757153	0	5.226757153	0	68.58125731	WARNING: Alcuni elementi Q_CD_req sono fuori dai limiti fisici
6	01:00:00	1	24.52886926	5.226757153	0	5.226757153	0	67.66747368	
7	01:15:00	1	24.52886926	5.226757153	0	5.226757153	0	67.07054675	
8	01:30:00	1	24.52886926	5.226757153	0	5.226757153	0	67.67302225	
9	01:45:00	1	24.52886926	5.226757153	0	5.226757153	0	66.86539434	
10	02:00:00	0	0	0	0	0	0	65.62287525	
11	02:15:00	0	0	0	0	0	0	60.52864676	
12	02:30:00	1	24.52886926	5.226757153	1.796600682	5.226757153	20.67215417	59.69712462	
13	02:45:00	0	0	0	0	0	0	62.35696363	
14	03:00:00	0	0	0	0	0	0	59.7494451	

Distretto | SOLVER | accumulatore_1 | conf_HOT2_1 | gen_foto_1 | gen_eoli_1 | carico_L1_1

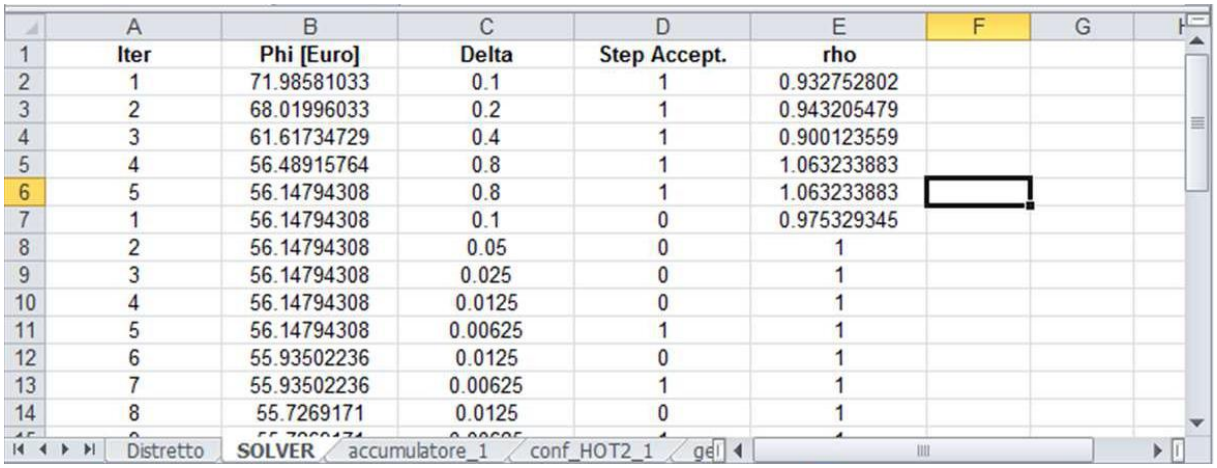
Figure 1.5.2: *Output: optimized CHP set points and related physical variables*



	A	B	C	D	E	F	G
1	Time	Phi(i) [Euro]	cW [Euro/kW]	W [kW]			
2	00:00:00	0.814458128	-0.03923162	-7.625075005			
3	00:15:00	0.338373347	-0.025	7.077629542			
4	00:30:00	0.338873858	-0.025	7.057609088			
5	00:45:00	0.823339503	-0.03923162	-7.85145808			
6	01:00:00	0.340814249	-0.025	6.979993455			
7	01:15:00	0.33682451	-0.025	7.139583007			
8	01:30:00	0.773921478	-0.036959387	-6.997069386			
9	01:45:00	0.340284195	-0.025	7.001195617			
10	02:00:00	0.619666143	-0.03623834	-17.09973861			
11	02:15:00	1.197318453	-0.03623834	-33.04010201			
12	02:30:00	0.723279628	-0.025	6.863919279			
13	02:45:00	0.618679184	-0.03623834	-17.07250337			
14	03:00:00	1.159044933	-0.035939693	-32.24971671			

Distretto | SOLVER | accumulatore_1 | conf_HOT2_1 | ge

Figure 1.5.3: *Output: cost of energy and power exchanged with the grid for each time unit*



	A	B	C	D	E	F	G
1	Iter	Phi [Euro]	Delta	Step Accept.	rho		
2	1	71.98581033	0.1	1	0.932752802		
3	2	68.01996033	0.2	1	0.943205479		
4	3	61.61734729	0.4	1	0.900123559		
5	4	56.48915764	0.8	1	1.063233883		
6	5	56.14794308	0.8	1	1.063233883		
7	1	56.14794308	0.1	0	0.975329345		
8	2	56.14794308	0.05	0	1		
9	3	56.14794308	0.025	0	1		
10	4	56.14794308	0.0125	0	1		
11	5	56.14794308	0.00625	1	1		
12	6	55.93502236	0.0125	0	1		
13	7	55.93502236	0.00625	1	1		
14	8	55.7269171	0.0125	0	1		

Distretto | SOLVER | accumulatore_1 | conf_HOT2_1 | ge

Figure 1.5.4: *Output: details of the optimization process related to the solver*

Chapter 2

Optimality conditions and Penalty functions

In this chapter we introduce some basic definitions of the optimization theory and we derive a mathematical characterization of the solutions of a constrained optimization problem. Then, we introduce some basic concepts of the penalty functions theory as we have employed them both in the PSO and in the SLP algorithm.

2.1 Optimality conditions

Consider the following nonlinear programming problem, subject to m equality constraints and p inequality constraints:

$$\min_x \ f(x), \tag{2.1.1a}$$

subject to

$$h_i(x) = 0 \qquad i \in \mathcal{E}, \tag{2.1.1b}$$
$$g_i(x) \leq 0 \qquad i \in \mathcal{I}. \tag{2.1.1c}$$

with $\mathcal{E} = \{1, \ldots, m\}$, $\mathcal{I} = \{1, \ldots, p\}$, and with $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^p$ smooth, real-valued functions.

We define the *feasible set* Ω for problem (2.1.1) to be the set of points x that satisfy all the constraints, that is [30],

$$\Omega = \{x | h_i(x) = 0, i \in \mathcal{E}; \ \ g_i(x) \leq 0, i \in \mathcal{I}\}, \tag{2.1.2}$$

$x \in \Omega$ is said to be a *feasible point*.

A vector x^* is a *local solution* of problem (2.1.1) if $x^* \in \Omega$ and it exists $r > 0$ such that $f(x) \geq f(x^*)$ for all $x \in B_r(x^*) \bigcap \Omega$, with $B_r(x^*) = \{x | \|x - x^*\|_2 \leq r\}$ a neighbourhood of x^*.

A vector x^* is a *global solution* of problem (2.1.1) if $x^* \in \Omega$ and $f(x) \geq f(x^*)$ for all $x \in \Omega$.

For a *convex problem*, i.e. a problem with convex objective function defined on a convex set, all local minima are also global minima. We recall that a set $\mathcal{C} \subset \mathbb{R}^n$ is said to be *convex* if the straight line segment connecting any two points in $\mathcal{C}$ lies entirely inside $\mathcal{C}$. Formally, for all $x, y \in \mathcal{C}$ and for all $t \in [0, 1]$, $tx + (1-t)y \in \mathcal{C}$. A function defined on a convex set $f : \mathcal{C} \to \mathbb{R}$, is said to be *convex* if for all $x, y \in \mathcal{C}$ and for all $t \in [0, 1]$:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y). \tag{2.1.3}$$

At a feasible point x, the inequality constraint $i \in \mathcal{I}$ is said to be *active* if $g_i(x) = 0$, and *inactive* if the strict inequality $g_i(x) < 0$ is satisfied. The *active set* $\mathcal{A}(x)$ at any feasible x is the set made of the indices of the active inequality constraints, that is:

$$\mathcal{A}(x) = \{i \in \mathcal{I} |\ g_i(x) = 0\}. \tag{2.1.4}$$

The *Lagrangian function* for the constrained optimization problem (2.1.1) is defined as

$$\mathcal{L}(x, \mu, \lambda) = f(x) - \sum_{i \in \mathcal{E}} \mu_i h_i(x) + \sum_{i \in \mathcal{I}} \lambda_i g_i(x), \tag{2.1.5}$$

where $\mu \in \mathbb{R}^m, \lambda \in \mathbb{R}^p$ are the *Lagrangian multipliers* for the equality and inequality constraints respectively, [30].

Definition 2.1 (LICQ)**.** Given the point x^* and the active set $\mathcal{A}(x^*)$ defined by (2.1.4), we say that the linear independence constraint qualification (LICQ) holds if the active constraints gradients $\{\nabla h_i(x^*),\ i \in \mathcal{E}; \nabla g_i(x^*),\ i \in \mathcal{A}(x^*)\}$ are linearly independent.

This condition allows us to state the following optimality conditions for a general nonlinear programming problem (2.1.1). These conditions are called first-order conditions because they concern themselves with properties of the gradients (first-derivative vectors) of the objective and constraint functions.

Theorem 2.1.1 (First-Order Necessary Conditions.)**.** *Suppose that x^* is a local solution of* (2.1.1) *and that the LICQ holds at x^*. Then there exist a Lagrange multiplier vector $\mu^* \in \mathbb{R}^m$, and a Lagrange multiplier vector $\lambda^* \in \mathbb{R}^p$, such that the following conditions are satisfied at (x^*, μ^*, λ^*):*

$$\begin{aligned}
\nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) &= 0, && && (2.1.6)\\
h_i(x^*) &= 0, && i \in \mathcal{E}, && (2.1.7)\\
g_i(x^*) &\leq 0, && i \in \mathcal{I}, && (2.1.8)\\
\lambda_i^* &\geq 0, && i \in \mathcal{I}, && (2.1.9)\\
\lambda_i^* g_i(x^*) &= 0, && i \in \mathcal{I}. && (2.1.10)
\end{aligned}$$

Conditions (2.1.6)-(2.1.10) are often known as the Karush–Kuhn–Tucker conditions, or KKT conditions for short. Since the *complementarity condition* (2.1.10) implies that the Lagrange multipliers corresponding to inactive inequality constraints are zero, we can omit the terms for indices $i \notin \mathcal{A}(x^*)$ from (2.1.6) and rewrite this condition as

$$0 = \nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in \mathcal{E}} \mu_i^* \nabla h_i(x^*) + \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla g_i(x^*). \tag{2.1.11}$$

For a given problem (2.1.1) and solution point x^*, there may be many vectors μ^*, λ^* for which the conditions (2.1.6)-(2.1.10) are satisfied. When the LICQ holds, however, the optimal μ^*, λ^* are unique, [30]. Notice that this conditions are only necessary, not sufficient. It is indeed easy to build an example in which conditions (2.1.6)-(2.1.10) are satisfied by a point that is not a local minimum, [30].

2.2 Penalty functions

Methods involving penalty functions seek the solution of a constrained optimization problem by replacing it by one or a sequence of unconstrained subproblems, [30]. The main idea of penalty

methods approach is to turn (2.1.1) to an unconstrained problem, incorporating the constraints in the original objective function, so that the objective function is replaced by a penalty function

$$\Phi(x, \nu) = f(x) + H(x, \nu) \tag{2.2.1}$$

that consists of

- the original objective of the constrained optimization problem f, plus
- a penalty term H that depends on a positive *penalty term* ν and consists of a sum of one additional term for each constraint, which is positive when the current point x violates that constraint and zero otherwise, and usually depends on the entity of constraint violation.

Then an unconstrained problem of the following form is obtained:

$$\min_x \Phi(x, \nu). \tag{2.2.2}$$

Various methods differ from each other for the choice of the penalty term H. Given a penalty function $\Phi(x, \nu)$, we assume that constraints violations are penalised increasingly by growing ν.

We can divide penalty methods into two big categories: methods based on *exact penalty functions* and the *exterior penalty methods*, [30].

Definition 2.2 (Exact Penalty Function)**.** A penalty function $\Phi(x, \nu)$ is said to be *exact* if there exists a positive scalar ν^* such that for any $\nu \geq \nu^*$, any local solution of the nonlinear programming problem (2.1.1) is a local minimizer of $\Phi(x, \nu)$.

So for all sufficiently large positive values of the penalty parameter ν, one minimization of unconstrained function $\Phi(x, \nu)$ yields a solution of the nonlinear programming problem (2.1.1). It is difficult to determine ν a priori in most practical applications however, usually a starting value is chosen and then the parameter is adjusted during the course of the computation, according to specific rules, [30].

An important example of exact penalty function is the l_1 function:

$$\Phi(x, \nu) = f(x) + \nu \sum_{i \in \mathcal{E}} |h_i(x)| + \nu \sum_{i \in \mathcal{I}} \max(0, g_i(x)). \tag{2.2.3}$$

Notice that l_1 penalty function is a non-smooth function.

It is possible to show that the l_1 function is an exact penalty function for all $\nu > \nu^*$, with

$$\nu^* = \max\{|\mu_i^*|, i \in \mathcal{E}; \quad \lambda_i^*, i \in \mathcal{I}\}. \tag{2.2.4}$$

For *exterior penalty methods* a single minimization of the penalty function Φ is not sufficient, but a sequence of penalized problems is solved:

$$\min_x \Phi(x, \nu_k) \quad \text{for} \quad k = 1, 2, \ldots. \tag{2.2.5}$$

The penalty functions $\Phi(x, \nu_k)$ depend on a positive penalty parameter ν_k that varies with iterations. By making this coefficient larger and larger, we penalize constraint violations more and more severely, thereby forcing the minimizer of the penalty function closer and closer to the feasible region for the constrained problem. Often, the minimizers of the penalty functions are infeasible with respect to the original problem, and approach feasibility only in the limit as the penalty parameter grows increasingly large. The simplest penalty function of this type

is the *quadratic penalty function*, in which the penalty terms are the squares of the constraint violations:

$$Q(x;\nu) = f(x) + \frac{1}{2}\nu \sum_{i\in\mathcal{E}} h_i^2(x) + \frac{1}{2}\nu \sum_{i\in\mathcal{I}} \max(0, g_i(x))^2 \tag{2.2.6}$$

where $\nu > 0$ is the penalty parameter. By driving ν to infinite, we penalize the constraint violations with increasing severity. So a sequence of values $\{\nu_k\}$ with $\nu_k \to \infty$ as $\kappa \to \infty$ is considered, and a approximate minimizer x_k of $Q(x, \nu_k)$ is sought, for each k.

A general framework for algorithms based on penalty function (2.2.6) can be specified as follows, [30].

Algorithm 2: Algorithm for quadratic penalty.

1. Set $k = 0$, specify the penalty parameter $\nu_0 > 0$, the tolerance $\tau_0 > 0$, the starting point x_0^s.
2. For $k = 0, 1, 2, \ldots$
   (a) Find an approximate minimizer x_k of $Q(\cdot, \nu_k)$, starting at x_k^s and terminating when $\|\nabla Q(x, \nu_k)\| \leq \tau_k$.
   (b) If a final convergence test is satisfied, stop with the approximate solution x_k. Otherwise, go to step 2c.
   (c) Choose a new penalty parameter:
   $$\nu_{k+1} \geq \nu_k,$$
   and a new starting point x_{k+1}^s and go to step 2a.

The convergence theory for Algorithm 2, allows wide latitude in the choice of tolerances τ_k, it requires only that $\lim_{k\to\infty} \tau_k = 0$, to ensure that the minimization is carried out more and more accurately.

Notice that when in problem (2.1.1) inequality constraints are present, the quadratic penalty function is a non-smooth function, while if only equality constraints are present the penalty function is a smooth function. The following two theorems, [30], describe some convergence properties of this approach when only equality constraints are present.

Theorem 1.
Suppose that each x_k is the exact global minimizer of $Q(\cdot, \nu_k)$ in Algorithm 2 above and that $\nu_k \to \infty$. Then every limit point x^* of the sequence $\{x_k\}$ is a solution of problem (2.1.1).

Since this result requires to find the global minimizer for each subproblem, its very desirable property of convergence to the global solution of (2.1.1) may be difficult to realize in practice. The next result concerns convergence properties of the sequence $\{x_k\}$ when we allow inexact (but increasingly accurate) minimizations of $Q(\cdot, \nu_k)$. In contrast to Theorem 1, it shows that the sequence is attracted to KKT points (that is, points satisfying first-order necessary conditions), rather than to a global minimizer, [30].

Theorem 2.
If the tolerance τ_k in Algorithm 2 above satisfies

$$\lim_{k\to\infty} \tau_k = 0 \tag{2.2.7}$$

and the penalty parameter satisfies $\nu_k \to \infty$, then for all limit points x^* of the sequence $\{x_k\}$ generated by Algorithm 2 at which the constraints gradients $\nabla h_i(x^*)$ are linearly independent,

we have that x^* is a KKT point for problem (2.1.1)*. For such points, we have for the infinite subsequence $\mathcal{K}$ such that $\lim_{k\in\mathcal{K}} x_k = x^*$ that*

$$\lim_{k\in\mathcal{K}} -h_i(x_k)\nu_k = \mu_i^*, \qquad \textit{for all } i \in \mathcal{E}, \tag{2.2.8}$$

where μ^* is multiplier vector that satisfies the KKT conditions.

Chapter 3

Sequential Linear Programming with penalty function

In this chapter we describe the SLP approach we devised in order to increase reliability of the SLP method originally implemented in ECOS ES and described in Section 1.3. We also provide theoretical convergence results of such a procedure.

3.1 Outline of the approach

In Section 1.3 we have shown that the SLP approach currently implemented in ECOS ES is not supported by any theoretical results and has the drawback that constraints of problem (1.3.5) may be not consistent, or the step provided may generate an infeasible solution approximation. The approach that is commonly used to deal with possible inconsistency of the constraints is to incorporate them in the form of a penalty term in the model objective. Following [5] and [19], we use an SLP trust region approach where the l_1 penalty function given in (2.2.3) is employed.

Here we focus on the solution of problem (1.2.1) and therefore we describe the algorithm and its convergence properties when applied to problems with only inequality constraints and bound constraints. The general outline of the algorithm we have implemented follows the lines of Algorithm 1, described in Section 1.3. The main difference concerns the LP subproblems we have to solve at each iteration. Here it is linearised the merit function Φ given in (2.2.3), as opposed to the function f. The function Φ in this case reduces to:

$$\Phi(x, \nu) = f(x) + \nu \sum_{i \in \mathcal{I}} \max(0, g_i(x)), \tag{3.1.1}$$

where we remind that $\mathcal{I} = \{1, \ldots, p\}$ is the set of indexes of inequality constraints. Its linear approximation at the current estimate x_k is given by:

$$l_k(d) = f(x_k) + \nabla f(x_k)^T d + \nu \sum_{i \in \mathcal{I}} \max(0, g_i(x_k) + \nabla g_i(x_k)^T d); \tag{3.1.2}$$

so that, at each iteration, we solve a subproblem of the form:

$$\min_d l_k(d) \tag{3.1.3a}$$

subject to

$$\max((x_{min} - x_k), -\bar{\Delta}_k) \leq d \leq \min((x_{max} - x_k), \bar{\Delta}_k), \tag{3.1.3b}$$

that has certainly a solution. The solution of problem (3.1.3), d_k, is used to compute the new solution approximation $x_{k+1} = x_k + d_k$. In the SLP approach described in Section 1.3 the step is

accepted only if x_{k+1} is feasible, here there is not a feasibility check and the step is accepted if it provides a sufficient decrease of the merit function, according to the rules described in Section 1.3. The ratio between the actual and the predicted reduction used to update the trust region radius here takes the following form:

$$\rho_k = \frac{\Phi(x_k) - \Phi(x_k + d_k)}{l_k(0) - l_k(d_k)} = \frac{\Delta\Phi_k}{\Delta l_k}. \tag{3.1.4}$$

For the practical implementation of this version of the algorithm we have to notice that function l_k is non-differentiable, but problem (3.1.3) can be written as the following equivalent smooth linear program, introducing the vector of slack variables t, [5]:

$$\min_{d,t} \nabla f(x_k)^T d + \nu \sum_{i \in \mathcal{I}} t_i; \tag{3.1.5a}$$

subject to

$$g_i(x_k) + \nabla g_i(x_k)^T d \leq t_i, \quad i \in \mathcal{I} \tag{3.1.5b}$$

$$\max((x_{min} - x_k), -\bar{\Delta}_k) \leq d \leq \min((x_{max} - x_k), \bar{\Delta}_k), \tag{3.1.5c}$$

$$t \geq 0. \tag{3.1.5d}$$

We now describe the outline of this version of SLP algorithm we have implemented for the specific problem (1.2.1) of energy districts. Values of fixed parameters are the same of Algorithm 1. The following algorithm sketches the k-th iteration of such a procedure.

Algorithm 3: k-th iteration of SLP algorithm with trust region and penalty function.

1. Given x_k, Δ_k, Δ_{max}, ν_k.
2. Evaluate $\nabla f(x_k)$ and $\nabla g_i(x_k)$ for $i = 1, \ldots, p$.
3. Solve the following LP:

$$\min_{d,t} \quad \nabla f(x_k)^T d + \nu_k \sum_{i \in \mathcal{I}} t_i \tag{3.1.6a}$$

   subject to:

$$g_i(x_k) + \nabla g_i(x_k)^T d \leq t_i, \qquad i \in \mathcal{I}, \tag{3.1.6b}$$

$$\max((x_{min} - x_k), -\bar{\Delta}_k) \leq d \leq \min((x_{max} - x_k), \bar{\Delta}_k), \tag{3.1.6c}$$

$$t \geq 0, \tag{3.1.6d}$$

   obtaining a candidate step d_k and multipliers estimates λ_k of (3.1.6b), π_k of (3.1.6c), $\bar{\lambda}_k$ of (3.1.6d).

4. Update the penalty parameter: if $\max\{\|\lambda_k\|_\infty, \|\bar{\lambda}_k\|_\infty\} > \nu_k$ then set $\nu_k = \max\{\|\lambda_k\|_\infty, \|\bar{\lambda}_k\|_\infty\}$.
5. Compute the step evaluation parameter:

$$\rho_k = \frac{\Phi(x_k) - \Phi(x_k + d_k)}{l_k(0) - l_k(d_k)}. \tag{3.1.7}$$

   Then:

(a) If $\rho_k \leq \rho_{bad}$, reduce the trust-region radius:

$$\Delta_{k+1} = \tfrac{1}{2}\Delta_k,$$

and go to step 6.

(b) ElseIf $\rho_k \geq \rho_{good}$, and in addition $\|d_k\|_\infty \geq 0.8\Delta_k$, increase the trust-region radius

$$\Delta_{k+1} = \min(2\Delta_k, \Delta_{max}).$$

Go to step 6.

(c) Else set $\Delta_{k+1} = \Delta_k$ and go to step 6.

6. If $\rho_k > \eta$ accept the step and set:

$$x_{k+1} = x_k + d_k.$$

Otherwise reject the step:

$$x_{k+1} = x_k.$$

Go to step 2.

3.2 Theoretical results

Following [19], it is possible to prove two important theoretical properties of this approach. Namely we can prove:

1. global convergence of the sequence $\{x_k\}$ generated by Algorithm 3 to a KKT point of $\Phi(x, \nu)$; we recall that if ν is sufficiently large, a minimizer of Φ is a solution of problem (1.2.1);

2. convergence of the multiplier estimates of problem (3.1.3) to the multipliers of problem (1.2.1).

We can express Φ in the following general compact form: $\Phi(x) = f(x) + H(g(x))$, where $H(g(x))$ is the penalty term and $g : \mathbb{R}^n \to \mathbb{R}^p$. The following results are indeed valid not only for l_1 penalty function, but for all polyhedral penalty convex functions H, [19]. Moreover, we prove the results in the most general case in which for the step $\|d_k\|$ a generic norm is considered, while, for seek of simplicity, bound constraints are not considered. So, as a consequence of the above assumptions, theoretical results are referred to the following unconstrained problem:

$$\min_x f(x) + H(g(x)), \tag{3.2.1}$$

and subproblem (3.1.3) becomes:

$$\min_d l_k(d) = f(x_k) + \nabla f(x_k)^T d + H(g(x_k) + J(x_k)^T d) \tag{3.2.2a}$$

subject to

$$\|d\| \leq \Delta_k, \tag{3.2.2b}$$

where $J(x) \in \mathbb{R}^{p\times n}$ is the Jacobian matrix of $g(x)$. Objective functions of problems (3.2.1) and (3.2.2) are not differentiable, so if we want to characterize stationary points of those problems we cannot use the first order necessary conditions introduced in Section 2.1. However it is possible to generalize the KKT conditions also to problems with non-smooth objective function. To do this we need the following definitions:

Definition 3.1. A vector v is a *subgradient* of a convex function f at $x \in \mathcal{D}$, where $\mathcal{D}$ is the domain of function f, if

$$f(y) \geq f(x) + v^T(y - x) \quad \text{for all} \quad y \in \mathcal{D}. \tag{3.2.3}$$

Definition 3.2. The *subdifferential* $\partial f(x)$ of f at x is the set of all subgradients:

$$\partial f(x) = \{v : v^T(y - x) \leq f(y) - f(x)\} \text{ for all} \quad y \in \mathcal{D}. \tag{3.2.4}$$

First order necessary KKT conditions for x^* to solve (3.2.1) are that there exists vectors of multipliers $\lambda^* \in \partial H(g^*)$ such that, (see Theorem 14.2.1 in [18]):

$$\nabla f(x^*) + J(x^*)\lambda^* = 0. \tag{3.2.5}$$

First order conditions for subproblem (3.2.2) are that there exist multipliers $\lambda_k \in \partial H(g(x_k)+ J(x_k)^T d_k)$, $w_k \in \partial \|d_k\|$ and $\pi_k \geq 0$, (see Theorem 14.6.1 in [18]), such that:

$$\nabla f(x_k) + J(x_k)^T \lambda_k + \pi_k w_k = 0, \tag{3.2.6}$$

$$\pi_k(\|d_k\| - \Delta_k) = 0. \tag{3.2.7}$$

The following Lemma is proved in [18] and is useful to prove the convergence theorems.

Lemma 3.2.1. *[Lemma 14.2.1 of [18]] Let $f : \mathcal{K} \to \mathbb{R}$ be a convex function, $\mathcal{K} \subset \mathbb{R}^n$ a convex set. Then*

- *$\partial f(x)$ is a closed convex set,*
- *$\partial f(x)$ is bounded for all $x \in B \subset \overset{o}{\mathcal{K}}$ where B is compact and $\overset{o}{\mathcal{K}}$ denotes the interior of $\mathcal{K}$.*

The next two Theorems are proved following the lines of the proofs of Theorem 2.1 and Theorem 2.2 in [19].

Theorem 3 (Global convergence of Algorithm 3)**.**
Let f and g be $\mathbb{C}^1$ functions and let $H(g)$ be a convex function. Let $\{x_k\}$ be the sequence generated by Algorithm 3. Either there exists an iteration index $\bar{k}$ such that $x_{\bar{k}}$ is a KKT point for $\Phi(x)$, or $\Phi(x_k) \to -\infty$ $k \to \infty$, or if the sequence $\{x_k\}$ is bounded, then there exists a subsequence S of indexes such that $\{x_k\}_{k\in S}$ has an accumulation point x^* which satisfies the KKT conditions for $\Phi(x)$, that is it exists a vector of multipliers λ^* such that:

$$\nabla f(x^*) + J(x^*)\lambda^* = 0. \tag{3.2.8}$$

Proof.
To prove the Theorem we need only consider the case that $\{\Phi_k\}$ is bounded below and $\{x_k\}$ is bounded. Because $\{x_k\}$ is bounded, there exists a subsequence S of iterations such that $\{x_k\}_{k\in S} \to x^*$. Suppose that:

a) d_k does not satisfy $\rho_k > \rho_{bad}$ for any $k \in S$ and $\{\Delta_k\}_{k\in S} \to 0$ and hence $\{\|d_k\|\}_{k\in S} \to 0$.

We define $\Delta\Phi_k = \Phi(x_k) - \Phi(x_k + d_k)$ and $\Delta l_k = l_k(0) - l_k(d_k) = \Phi(x_k) - l_k(d_k)$. A consequence of $\mathbb{C}^1$ continuity of f and g, convexity of $H(g)$ and boundedness of $\partial H(g)$, which follows from Lemma 3.2.1, and of the use of the first order Taylor expansion, is that:

$$\Delta\Phi_k = \Delta l_k + o(\|d_k\|) \tag{3.2.9}$$

and hence

$$\frac{\Delta\Phi_k}{\Delta l_k} \to 1 \qquad k \to \infty, \tag{3.2.10}$$

which contradicts the fact that $\rho_k = \frac{\Delta\Phi_k}{\Delta l_k} > \rho_{bad}$ fails for all $k \in S$. Therefore this case is inconsistent and it certainly exists a subsequence of indexes S such that:

b) d_k satisfies $\rho_k > \rho_{bad}$ and $\liminf_{k \in S} \Delta_k > 0$.

In fact, let S' be a sequence of indexes of unsuccessful iterations. If S' is finite we clearly have $\rho_k > \rho_{bad}$ for k sufficiently large. Otherwise suppose $k_0 \in S'$. Since $\{\Delta_k\}_{k \in S'} \nrightarrow 0$, otherwise we obtain case (a) again, for each $k_0 \in S'$ it exists i such that $k_0 + i$ is a successful iteration with $k_0 + i \notin S'$. Therefore $x_{k_0+i} = x_{k_0}$ and the subsequence $\{x_{k_0+i}\}_{k_0 \in S'} \to x^*$. If we let $S = \{k_0 + i, k_0 \in S'\}$, $\{x_k\}_{k \in S}$ is the subsequence of case (b). In case (b) it can be assumed that $\liminf_{k \in S} \Delta_k > \bar{\Delta} > 0$, as if $\liminf_{k \in S} \Delta_k = 0$ thus imply $\liminf \Delta_k = 0$ and this yields case (a) that we have proved to be inconsistent. Because $\Phi_1 - \Phi^* \geq \sum_{k \in S} \Delta\Phi_k$, it follows that $\sum_{k \in S} \Delta\Phi_k$ converges. Then, $\rho_k \geq \rho_{bad}$, i.e. $\Delta\Phi_k \geq \Delta l_k \rho_{bad}$, yields the convergence of the series $\sum_{k \in S} \Delta l_k$, and hence $\{\Delta l_k\} \to 0$.

Define $l^*(d) = f(x^*) + \nabla f(x^*)d + H(g(x^*) + J(x^*)^T d)$. Let

$$\bar{d} = \arg\min l^*(d) \tag{3.2.11a}$$

$$s.t. \|d\| \leq \bar{\Delta} \tag{3.2.11b}$$

and denote $\bar{x} = x^* + \bar{d}$. Then

$$\|\bar{x} - x_k\| \leq \|\bar{x} - x^*\| + \|x^* - x_k\| = \|\bar{d}\| + \|x^* - x_k\|. \tag{3.2.12}$$

Therefore,

$$\|\bar{x} - x_k\| \leq \bar{\Delta} + \|x^* - x_k\| \leq \Delta_k \tag{3.2.13}$$

for all k sufficiently large, $k \in S$. Thus $\bar{x}$ is feasible for problem (3.2.2), so

$$l_k(\bar{x} - x_k) \geq l_k(d_k) = \Phi(x_k) - \Delta l_k. \tag{3.2.14}$$

In the limit, for $k \in S$, $\nabla f(x_k) \to \nabla f(x^*), g(x_k) \to g(x^*), J(x_k) \to J(x^*), \bar{x} - x_k \to \bar{d}$, and $\Delta l_k \to 0$, so it follows that $l^*(\bar{d}) \geq \Phi(x^*) = l^*(0)$. Thus $d = 0$ also minimizes $l^*(d)$ subject to $\|d\| \leq \bar{\Delta}$, and since the latter constraint is not active it follows from (3.2.7) that $\pi^* = 0$ and from (3.2.6) that it exists λ^* such that $\nabla f(x^*) + J(x^*)\lambda^* = 0$, then x^* is a KKT point. □

Theorem 3 states the global convergence of Algorithm 3, i.e. in case the objective function is not unbounded, it states the existence of an accumulation point x^* of the sequence $\{x_k\}$ generated by Algorithm 3, regardless of the starting solution approximation, that satisfies KKT conditions (3.2.8) for $\Phi(x, \nu)$. In Section 3.3 we will describe the penalty parameter update strategy we have chosen, which ensures that the penalty parameter ν is large enough to let x^* be a solution of the original problem (2.1.1).

Theorem 4 (Convergence of multipliers)**.** *Let f and g be $\mathbb{C}^1$ functions, $H(g)$ a convex function, π_k and λ_k multipliers of subproblems* (3.2.2)*, defined in equations* (3.2.6) *and* (3.2.7)*. If the subsequence S in the statement of Theorem 3 exists, then $\{\pi_k\}_{k \in S} \to 0$. Moreover any accumulation point λ^* of the multiplier vectors λ_k, $k \in S$, satisfies $\lambda^* \in \Lambda^*$, where $\Lambda^* = \{\lambda : \lambda$ satisfies KKT conditions at $x^*\}$, and such an accumulation point exists.*

Proof. The definition of $l_k(d)$ and the subgradient inequality (3.2.3) yields

$$\Delta l_k(d_k) = -\nabla f(x_k)^T d_k + H(g(x_k)) - H(g(x_k) + J(x_k)^T d_k) \tag{3.2.15}$$

$$\geq -\nabla f(x_k)^T d_k - \lambda_k^T J(x_k)^T d_k. \tag{3.2.16}$$

It follows from (3.2.6) that

$$\Delta l_k(d_k) \geq \pi_k w_k^T d_k = \pi_k \|d_k\|, \tag{3.2.17}$$

where the last equality follows from the fact that

$$w_k \in \partial\|d_k\| = \{\lambda : \lambda^T d_k = \|d_k\|\}, \tag{3.2.18}$$

from equation 14.3.7 of [18].

From the proof of Theorem 3 it follows that for the subsequence S, satisfying assumptions (b), $\{\Delta l_k\}_{k\in S} \to 0$ and hence $\{\pi_k\|d_k\|\}_{k\in S} \to 0$ from (3.2.17). From this it can be deduced that $\{\pi_k\}_{k\in S} \to 0$ as follows. Conversely let $\pi_k \geq \beta > 0$ on some subsequence $S' \subset S$. It follows that $\{d_k\}_{k\in S'} \to 0$ and also from (3.2.7) that $\|d_k\| = \Delta_k$ for $k \in S'$, but these conditions contradict $\Delta_k > \bar{\Delta} > 0$ in case (b) thus $\{\pi_k\}_{k\in S} \to 0$. Now consider the sequence $\{\lambda_k\}_{k\in S}$. Because $\{x_k\}_{k\in S} \to x^*$ and $\Delta_k \leq \Delta_{max}$, it follows that the vectors d_k and $g(x_k)+J(x_k)^T d_k$ are bounded. Existence of an accumulation point of sequence $\{\lambda_k\}_{k\in S}$ is a consequence of Lemma 3.2.1. In fact Lemma 3.2.1 yields that ∂H is compact and $\{\lambda_k\}_{k\in S} \in \partial H$, and a subsequence in a compact has an accumulation point. Let $\{\lambda_k\}_{k\in S'} \to \lambda^*$. In the limit it now follows from (3.2.6) and $\{\pi_k\}_{k\in S} \to 0$ that

$$\nabla f(x^*) + J(x^*)\lambda^* = 0. \tag{3.2.19}$$

Moreover the subgradient inequality and $\lambda_k \in \partial H(g(x_k) + J(x_k)^T d_k)$ give that, for all $g \in \mathbb{R}^n$

$$\begin{aligned} H(g) &\geq H(g(x_k) + J(x_k)^T d_k) + (g - g(x_k) - J(x_k)^T d_k)^T \lambda_k && (3.2.20)\\ &= H(g(x_k)) - \Delta l_k - \nabla f(x_k)^T d_k + (g - g(x_k) - J(x_k)^T d_k)^T \lambda_k && (3.2.21)\\ &= H(g(x_k)) - \Delta l_k - (g - g(x_k))^T \lambda_k + \pi_k\|d_k\| && (3.2.22)\\ & && (3.2.23) \end{aligned}$$

using the definition of Δl_k and (3.2.6). In the limit as $k \to \infty$, $\Delta l_k \to 0$, $g(x_k) \to g(x^*)$, $H(g(x_k)) \to H(g(x^*))$, $\lambda_k \to \lambda^*$ and $\pi_k\|d_k\| \to 0$, so it follows that

$$H(g) \geq H(g(x^*)) + (g - g(x^*))^T \lambda^* \qquad \forall g \in \mathbb{R}^n, \tag{3.2.24}$$

that is $\lambda^* \in \partial H(x^*)$. Together with (3.2.19) we see that λ^* satisfies KKT conditions at x^*. □

Theorem 4 states that, if the subsequence S of Theorem 3 exists, than the subsequence of multipliers of subproblems (3.2.2) approximates multipliers of the original problem. We have used this important theoretical result for the stopping criterion of Algorithm 3, that we describe in the following section.

3.3 Implementation issues

In this section we discuss two important features of the algorithm:

- the stopping criterion,
- the updating strategy for the penalty parameter.

As far as the stopping criterion is concerned, our aim is to provide a measure of the closeness of the current iterate to a KKT point of the original problem. We have followed the approach suggested in [5], that is we use the following stopping criterion. We stop Algorithm 3 whenever a pair (x_k, λ_k) satisfies the following conditions:

$$\max\{\|\nabla f(x_k) + J(x_k)^T \lambda_k\|_\infty, \|g(x_k)^T \lambda_k\|_\infty\} < \epsilon(1 + \|\lambda_k\|_2); \tag{3.3.1}$$

$$\max\{\max_{i\in\mathcal{I}}(0, g_i(x_k)), \max(0, x_{min} - x_k), \max(0, x_k - x_{max})\} < \epsilon(1 + \|x_k\|_2) \tag{3.3.2}$$

with ϵ a tolerance to be fixed and λ_k the Lagrange multiplier vector corresponding to the inequality constraints (3.1.6b). We can use this criterion as we are supported by Theorem 4,

that states that the multipliers λ_k of the LP subproblems (3.1.3) estimate the multipliers of the original problem. Note that we do not have multipliers of (3.1.3) at disposal, and the only informations we can obtain are the estimates of the multipliers of the smooth subproblems (3.1.6) we are actually solving at each iteration k.

Comparing KKT conditions of problems (3.1.3) and (3.1.6), we can prove that they are equivalent. For seek of simplicity, we omit the contribution of bound constraints.

The Lagrangian function for problem (3.1.6) is:

$$\mathcal{L}(x, \lambda_k, \bar{\lambda}_k) = \nabla f(x_k)^T d + \nu_k \sum_{i \in I} t_i - \sum_{i \in I} t_i \bar{\lambda}_{k,i} + \sum_{i \in I} (\nabla g_i(x_k)^t d - t_i + g_i(x_k)) \lambda_{k,i} +, \\ + \sum_{i \in I} \pi_{2k,i} (d - \Delta_k)_i + \sum_{i \in I} \pi_{1k,i} (-d - \Delta_k)_i. \tag{3.3.3}$$

We denote with $\bar{\nu}_k$ a p dimensional vector with all components equals to ν_k. KKT conditions for this problem are:

$$\nabla f(x_k) + J(x_k)^T \lambda_k + \pi_{2k} - \pi_{1k} = 0; \tag{3.3.4}$$
$$\bar{\nu}_k - \lambda_k - \bar{\lambda}_k = 0; \tag{3.3.5}$$
$$t \geq 0, \tag{3.3.6}$$
$$\nabla g_i(x_k)^T d - t_i + g_i(x_k) \leq 0, \text{ for } i \in I, \tag{3.3.7}$$
$$-\Delta_k - d \leq 0 \tag{3.3.8}$$
$$d - \Delta_k \leq 0 \tag{3.3.9}$$
$$\lambda_k \geq 0 \tag{3.3.10}$$
$$\bar{\lambda}_k \geq 0 \tag{3.3.11}$$
$$\pi_{1k} \geq 0 \tag{3.3.12}$$
$$\pi_{2k} \geq 0 \tag{3.3.13}$$
$$(J(x_k)^T d - t + g(x_k))^T \lambda_k = 0, \tag{3.3.14}$$
$$t^T \bar{\lambda}_k = 0, \tag{3.3.15}$$
$$(-\Delta_k - d)^T \pi_{1k} = 0 \tag{3.3.16}$$
$$(d - \Delta_k)^T \pi_{2k} = 0. \tag{3.3.17}$$

Firsts two conditions can be put together obtaining:

$$\nabla f(x_k) + J(x_k)^T (\bar{\nu}_k - \bar{\lambda}_k) + \pi_{2k} - \pi_{1k} = 0. \tag{3.3.18}$$

Considering that we have chosen ∞-norm for trust region constraints, equations (3.3.4)-(3.3.17) are exactly the KKT conditions (3.2.6), (3.2.7) of problem (3.1.3).

Then, we can use the approximate multipliers of (3.1.6), provided by the function used to solve the LPs, in (3.3.1), (3.3.2).

The stopping criterion (3.3.1), (3.3.2) gives us a measure of the closeness of the computed solution to a stationary point of the original problem (1.2.1).

As far as the penalty parameter is concerned, we would like to have penalty function (3.1.1) to be exact, according to Definition 2.2 and Equation (2.2.4). Then, in Algorithm 3 the following updating strategy is adopted: if $\nu_k < \max\{\|\lambda_k\|_\infty, \|\bar{\lambda}_k\|_\infty\}$ then $\nu_k = \max\{\|\lambda_k\|_\infty, \|\bar{\lambda}_k\|_\infty\}$ is set, where we have used current Lagrange multiplier estimates as an approximation to multipliers of the original problem λ^*, that should be used for the computation of parameter ν^*.

Chapter 4

Particle swarm optimization

SLP algorithm described in Section 1.3, that Enel equipped ECOS ES with, is shown to converge only to a local minimum, as well as the SLP approach described in Chapter 3. Then, the research centre Enel Ingegneria e Ricerca decided to study and implement also methods of different type that provide convergence to a global minimum of an optimization problem, as opposed to the available solver. Among the various procedures available in the literature, the Particle Swarm Optimization (PSO) method was chosen. In this chapter we introduce PSO method and describe its main features. Namely we describe the main ideas characterizing the approach, how to choose free coefficients in the equations defining the method, various topology schemes, some procedures to prevent the swarm to be stuck in local minima. As far as the ECOS project is concerned, we have devised a specific version of PSO algorithm suitable for the specific problem of optimization of energy districts (1.2.1). We have implemented it in Matlab and made it available into the ECOS ES code, as an alternative to the already available SLP algorithm.

Classical methods as SLP previously described generates sequences converging to a local minimum of an optimization problem. Moreover the objective function is required to be differentiable and its first order derivative must be available. When a global minimizer is needed and/or the objective function is non smooth or its gradient is not available different approaches must be used. A wide class of methods have been proposed taking inspiration from several systems observed in nature, and adapting mechanisms of those natural systems, [23], [31]. These methods are based on stochastic algorithms, i.e. methods that use random variables as opposed to deterministic algorithms that, given a particular input, will always produce the same output because results on each step are completely determined by previous computation. Such algorithms evolve a population of interacting agents that perform a search in the domain space. The main advantage of using these approaches is that they require only function evaluations and first order derivative are not needed. These methods indeed are derivative free and use only informations gained by function evaluations, so that they can be used to deal with optimization problems with a non-smooth objective function. Although these optimization approaches are not supported by theoretical results of convergence, their promising results on complex problems, and overall on problems with strong nonlinearities, offered a boost to research. Research groups attained to refine early variants of these algorithms, introducing a set of efficient approaches under the general name of **Evolutionary Algorithms**, that are now considered promising alternatives in cases where classical approaches were not applicable, [31].

Ones of the most famous among these approaches are the **Genetic Algorithms** (GA), developed in the 50-60's, [20], [23]. These are search heuristic that mimics the process of natural selection and generate solutions to optimization problems using techniques, such as inheritance, mutation, selection, and crossover, inspired by natural evolution and to the Darwinian biological theory. In a genetic algorithm, a population of candidate solutions (called individuals, creatures,

or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation, so only the best individuals survive while the others are rejected. The new generation of candidate solutions is then used in the next iteration of the algorithm, [23].

The success recognized by evolutionary approaches sparked off research all over the world. As a result, in the 90's a new category of algorithms appeared. This kind of algorithms exploit a different approach from the GA, instead of modelling evolutionary procedures in microscopic (DNA) level, these methods model populations in a macroscopic level, i.e. in terms of social structures and aggregating behaviours. Hierarchically organized societies of simple organisms, such as ants, bees, and fish, with a very limited range of individual responses, exhibit fascinating behaviours with identifiable traits of intelligence as a whole. In fact although each agent has a very limited action space and there is no central control, the aggregated behaviour of the whole swarm exhibits emergent properties that cannot be described simply by aggregating the behaviour of each team member, i.e. an ability to react to environmental changes and decision making capacities. The lack of a central tuning and control mechanism in such systems has triggered scientific curiosity. Simplified models of these systems were developed and studied through simulations and a new branch of artificial intelligence emerged, under the name of **Swarm Intelligence**, [31]. Swarm intelligence studies the collective behaviour and emergent properties of complex, self-organized, decentralized systems with social structure.

Particle Swarm Optimization (PSO) is an optimizer for nonlinear functions that was introduced in 1995 by James Kennedy and Russell C. Eberhart, [12]. They took inspiration by previous works of scientists that have created computer simulations of the movement of organisms in a bird flock or fish school. Notably, Reynolds and Heppner [34] and Grenander [21] presented simulations of bird flocking. Reynolds was intrigued by the aesthetics of bird flocking choreography, and Heppner, a zoologist, was interested in discovering the underlying rules that enabled large numbers of birds to flock synchronously, often changing direction suddenly, scattering and regrouping. Both of these scientists had the insight that local processes might underlie the unpredictable group dynamics of bird social behaviour, and that something about the flock dynamic enables members of the flock to capitalize on one another's knowledge. As sociobiologist E. 0. Wilson [37] has written, in reference to fish schooling: "in theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches". This statement suggests that social sharing of information among conspeciates offers an evolutionary advantage: this hypothesis was fundamental to the development of Particle Swarm Optimization. In fact this approaches enhance a cooperative attitude among different individuals rather than a competitive one, as in GA, and it does not practice any selection with the idea that less successful individuals at present time may become the most successful ones, [27].

Following the natural metaphor, PSO evolves a population of individuals, referred to as particles, within the search space, that behave according to simple rules but interact to produce a collective behaviour. Each individual flies through the search space by updating its individual velocity toward both the best position or location it personally has found (i.e. the personal best), and toward the globally best position found by the entire swarm (i.e. the global best). Sharing

the globally personal best position found models the social act of communication.

In optimization problems an individual represents a possible solution of the problem, and a fitness function is used to evaluate the population so that the best particle is the one that offers the lowest (for a minimization problem) function value.

In many complex real life problems PSO approach shows to perform better than other Evolutionary Algorithms, such as GA. For a wide overview see [27].

4.1 Outline of the algorithm

PSO can be used to solve optimization problems of the following form:

$$\min_x f(x) \tag{4.1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a function on which it is not necessary to make regularity assumptions, n is the dimension of the search space and $x \in \mathbb{R}^n$,[31].

Assume to have a swarm constituted by s particles. The i-th particle of the swarm is identified by three different d-dimensional vectors:

- x^i, position vector of the i-th particle;
- v^i, velocity vector of the i-th particle;
- p^i_{best}, vector of the best position achieved so far by the i-th particle.

There is also another vector necessary for the optimization process, p^g_{best}, that represents the best position achieved so far by the swarm.

PSO are iterative processes. At first iteration positions of particles are randomly initialized within the search space. Let us denote by $x^i_k \in \mathbb{R}^n$ the position of the i-th particle at iteration k. The initial guess x^i_0 is set as follows:

$$x^i_0 = x_{min} + \bar{r}^i(x_{max} - x_{min}) \tag{4.1.2}$$

for $i = 1, \ldots, s$, where $\bar{r}^i$ are d-dimensional vectors of pseudo-random numbers with components selected from uniform distribution, $\bar{r}^i \sim U(0, 1)$. Velocities vectors are initialized to zero, p^i_{best} are set to current positions, p^g_{best} is the best of all initial positions.

At current iteration k the swarm is evolved using the following update rules:

$$v^i_{k+1} = wv^i_k + c_1 r_1(p^i_{best,k} - x^i_k) + c_2 r_2(p^g_{best,k} - x^i_k); \tag{4.1.3}$$

$$x^i_{k+1} = x^i_k + v^i_{k+1}; \tag{4.1.4}$$

where x^i_k and v^i_k are position and velocity vectors of the i-th particle at k-th iteration, c_1 and c_2 are positive weighting factors of *cognitive* and *social* components respectively, r_1 and r_2 are random variables uniformly distributed within $[0, 1]$, w is the inertia weight.

Referring to equation (4.1.3), it consists of three different parts:

- wv^i_k is the contribution of the previous velocity to the new one;
- $c_1 r_1(p^i_{best} - x^i)$ is the *cognitive* contribution, each particle remembers the best position ever visited and tend to return to it;
- $c_2 r_2(p^g_{best} - x^i)$ is the *social* contribution, each particle knows the best position achieved by the entire swarm and is attracted by it.

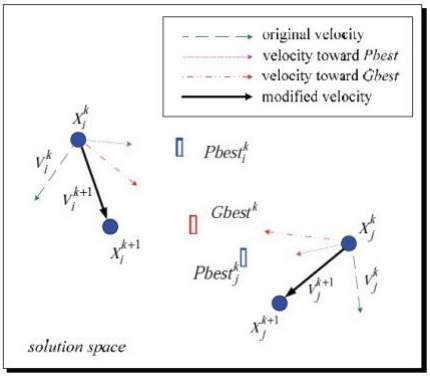


Figure 4.1.1: *Particles velocity updating scheme. Here we denote with x_i^k, v_i^k and $Pbest_i^k$ the position, velocity and p_{best} vectors, respectively, of the i-th particle at k-th iteration. $Gbest^k$ is $p_{best,k}^g$, i.e. vector p_{best}^g at k-th iteration.*

The three contributions are depicted in Figure 4.1.1. Setting the three parameters c_1, c_2, w is possible to decide how to weight each contribution. This choice is really important because it affects velocity convergence and PSO's performance. Is not so easy to choose right parameters, because of a strong dependence on the specific problem, so a numerical experimentation is necessary, even if it is possible to follow some general rules that we describe in the following sections. After the update of the swarm through equations (4.1.3) and (4.1.4), the objective function is evaluated for each particle and those values are used to update best positions. Thus, the new best position of x^i at iteration $k+1$ is defined as follows:

$$p_{best,k+1}^i = \begin{cases} x_k^i & \text{if } f(x_k^i) < f(p_{best,k}^i) \\ p_{best,k}^i & \text{otherwise} \end{cases} \tag{4.1.5}$$

The global best is updated too. Letting $p = \arg\min_{i=1\dots s} f(p_{best,k+1}^i)$,

$$p_{best,k+1}^g = \begin{cases} p & \text{if} f(p) < f(p_{best,k}^g) \\ p_{best,k}^g & \text{otherwise} \end{cases} \tag{4.1.6}$$

We here describe a scheme of the k-th iteration of the basic PSO algorithm:

Algorithm 4: k-th iteration of PSO algorithm.

1. Given c_1, c_2, w, x_k^i, v_k^i, $p_{best,k}^i$, $p_{best,k}^g$ for $i = 1, \dots, s$, perform the following steps.
2. Calculate $r_1, r_2 \sim U(0,1)$ and evolve the swarm:
$$v_{k+1}^i = w v_k^i + c_1 r_1 (p_{best,k}^i - x_k^i) + c_2 r_2 (p_{best,k}^g - x_k^i); \tag{4.1.7}$$
$$x_{k+1}^i = x_k^i + v_{k+1}^i; \tag{4.1.8}$$
for $i = 1, \dots, s$.
3. Evaluate the objective function of each particle of the swarm and update vectors $p_{best,k+1}^i$:
$$p_{best,k+1}^i = \begin{cases} x_k^i & \text{if } f(x_k^i) < f(p_{best,k}^i) \\ p_{best,k}^i & \text{otherwise} \end{cases} \tag{4.1.9}$$
and $p_{best,k+1}^g$:
$$p_{best,k+1}^g = \begin{cases} p & \text{if} f(p) < f(p_{best,k}^g) \\ p_{best,k}^g & \text{otherwise} \end{cases} \tag{4.1.10}$$
where $p = \arg\min_{i=1\dots s} f(p_{best,k+1}^i)$.

4.2 Choice of free parameters: stability analysis

In this section we perform a stability analysis of the dynamical system defined by updating equations (4.1.3), (4.1.4) that is useful for the choices of free parameters.

In order to have the PSO algorithm to perform well, a fine tuning of free parameters is necessary. There are three parameters to be set: the inertia weight w that weights the effect of the memory of previous velocity on the new one, c_1 and c_2 that weight the cognitive and the social components respectively. Without the last two terms particles will keep on "flying" at the current speed in the same direction until they hit the boundary. In more details, if $c_1 > 0$ and $c_2 = 0$ particles don't communicate among them and perform a local search, while if $c_1 = 0$ and $c_2 > 0$ all the particles are attracted from a single point and the exploring capacity is really low. On the other hand if the first term is not present the search space statistically shrinks through the generations and PSO resembles a local search algorithm [15]. The choice of these coefficients is really important because it affects not only the convergence velocity, but also the quality of the final solution. There are no general rules for this choice, but the stability analysis of the dynamic nonlinear system defined by the swarm it is useful to restrict the choice to a bounded region. Once the stability region is detected, a choice of free parameters inside this region ensures the swarm to convergence to a single point. Notice that there are no theoretical results ensuring that this point is the global minimum, PSO is indeed an heuristic algorithm. For the theoretical analysis of PSO we follows the lines of the procedure described in [36]. For this purpose the deterministic version of updating equations, obtained by setting random numbers to their expected values: $r_1 = r_2 = \frac{1}{2}$, will be considered. The exact relationship between the random and the deterministic versions of the algorithm is not yet rigorously been established, but in practise the contribution of random variables seems to enhances the zigzagging tendency and slows down convergence, thus improving the state space exploration, [36]. The algorithm description can be reduced for analysis purposes to the one-dimensional case, without loss of generality, because each dimension is updated independently from the others. For seek of simplicity we will omit the superscript i and we will denote with x_k the position of a particle. To perform the stability analysis, we write updating equations in the following general form:

$$v_{k+1} = av_k + b_1\frac{1}{2}(p^i_{best,k} - x_k) + b_2\frac{1}{2}(p^g_{best,k} - x_k); \tag{4.2.1}$$

$$x_{k+1} = cx_k + dv_{k+1}. \tag{4.2.2}$$

We will denote $p_{1,k} = p^i_{best,k}$, $p_{2,k} = p^g_{best,k}$. Equations (4.2.1) and (4.2.2) could be simplified using the notation:

$$b = \frac{b_1 + b_2}{2}; \tag{4.2.3}$$

$$p_k = \frac{b_1}{b_1 + b_2}p_{1,k} + \frac{b_2}{b_1 + b_2}p_{2,k}. \tag{4.2.4}$$

Thus we obtain:

$$v_{k+1} = av_k + b(p_k - x_k); \tag{4.2.5}$$

$$x_{k+1} = cx_k + dv_{k+1}. \tag{4.2.6}$$

The algorithm described by equations (4.2.5) and (4.2.6) contains four tuning parameters a, b, c and d. It will be now shown that only two of them are truly useful, while the other two can be fixed arbitrarily without loss of generality, obtaining (4.1.3) and (4.1.4) again. Using equations (4.2.5) and (4.2.6) the velocity can be eliminated from the description of the algorithm yielding the following second order recursion formula involving only successive particle positions:

$$x_{k+1} + (bd - a - c)x_k + acx_{k-1} = bdp_k. \tag{4.2.7}$$

Indeed, substituting (4.2.5) in (4.2.6):

$$x_{k+1} = cx_k + d(av_k + b(p_k - x_k)) = cx_k + adv_k + dbp_k - dbx_k. \tag{4.2.8}$$

Then, from (4.2.6) $dv_k = x_k - cx_{k-1}$, and substituting it in (4.2.8) ones obtains (4.2.7):

$$x_{k+1} = cx_k + ax_k - acx_{k-1} + dbp_k - dbx_k = (c + a - db)x_k - acx_{k-1} + dbp_k. \tag{4.2.9}$$

It appears that individual values of coefficients b and d are not important; the only important quantity is the product bd. Without any loss of generality, one can always set, for example, $d = 1$. In other words, any sequence of particle positions $\{x_k\}$ generated by the PSO algorithm described by equations (4.2.5) and (4.2.6) can also be generated with $d = 1$ and a suitably chosen value of b. The sequence of $\{v_k\}$ will be different, however, but this has no impact on the optimization algorithm since the objective function only depends on x, with v being just an auxiliary variable, [36].

Notice that p_k in (4.2.4) depends on iteration index k, but at equilibrium particles do not find better positions so p_1, p_2 and hence p_k do not change, so we can assume them to be constant, and set $p_k = p$, [36]. For optimization purposes it is desired that, in the long run, the population of particles converges to the optimum location found so far:

$$\lim_{k\to\infty} x_k = p. \tag{4.2.10}$$

In order to have the sequence defined by (4.2.7) to converge it is necessary and sufficient for the characteristic polynomial associated to (4.2.7)

$$p_c(z) = z^2 + (b - a - c)z + ac \tag{4.2.11}$$

to be a Schur polynomial, i.e. all the roots have magnitude less than 1, [4]. It exists a practical criterion, the *First Schur criterion*, that can be used to find conditions on coefficients that guarantee the desired property. To introduce it, we need the following definitions, [4].

Definition 4.1. Given a polynomial of degree n, $\rho(z) = \sum_{i=0}^{n} \rho_i z^{n-i}$, the *adjunct polynomial* is defined as: $q(z) = \sum_{i=0}^{n} \bar{\rho}_i z^i$, where $\bar{\rho}_i$ are the complex conjugates of coefficients ρ_i.

Definition 4.2. Given a polynomial $\rho(z)$, and his adjunct polynomial $q(z)$, the *reduced polynomial* is defined as:

$$\rho^{(1)}(z) = \frac{q(0)\rho(z) - \rho(0)q(z)}{z}.$$

Theorem 4.2.1 (First Schur criterion)**.** *$\rho(z) = \rho_0 z^n + \rho_1 z^{n-1} + \cdots + \rho_n$ is a Schur polynomial if and only if the two following conditions hold:*

- $|\rho_0| > |\rho_n|$*;*
- *the reduced polynomial $\rho^{(1)}(z)$ is a Schur polynomial.*

The sequence defined by (4.2.7) has an equilibrium point at:

$$\hat{z} = \frac{bp}{1 + b - a - c + ac}. \tag{4.2.12}$$

A necessary condition to have p as the equilibrium point is:

$$\frac{b}{1 + b - a - c + ac} = 1, \tag{4.2.13}$$

and so

$$(a-1)(c-1)=0. \tag{4.2.14}$$

The choices $a=1$ or $c=1$ are equivalent as far as the sequence of particle positions x_k is concerned because equation (4.2.7) is symmetric with respect to a and c. Usually the case $c=1$ is considered, because the choice $c=d=1$ has the nice feature that the variable v can be interpreted as a true "velocity ", i.e. the difference between two successive particle positions. So if we want our sequence to converge to p, i.e. to have p as a stable equilibrium point, we can now apply the Schur criterion to the polynomial:

$$p_c(z)=z^2+(b-a-1)z+a. \tag{4.2.15}$$

The first condition of Theorem 4.2.1 is:

$$|a|<1, \text{ i.e. } -1<a<1. \tag{4.2.16}$$

To check the second condition we need to compute the adjunct polynomial:

$$q(z)=az^2+(b-a-1)z+1; \tag{4.2.17}$$

and the reduced polynomial:

$$\rho^{(1)}(z) = \frac{(z^2+(b-a-1)z+a)-a(az^2+(b-a-1)z+1)}{z}= \tag{4.2.18}$$

$$= (1-a^2)z+(1-a)(b-a-1). \tag{4.2.19}$$

The reduced polynomial is a first degree polynomial, so we can check if it is a Schur polynomial by simply computing its root. Note that from (4.2.16) $a\neq 1$, so root of polynomial (4.2.19) is $z=\frac{1+a-b}{1+a}$. Then, the second condition of Theorem 4.2.1 is:

$$\left|\frac{1+a-b}{1+a}\right|<1. \tag{4.2.20}$$

Taking into account that from (4.2.16) $1+a>0$, (4.2.20) becomes:

$$\begin{cases} b>0 & \text{if } 1+a-b\geq 0 \\ 2a-b+2>0 & \text{if } 1+a-b<0 \end{cases}. \tag{4.2.21}$$

The stability region is the region in the (a,b) plane in which conditions (4.2.16) and (4.2.21) hold simultaneously. It is the triangle depicted in Figure 4.2.1.

For any initial position and velocity, the particle will converge to its equilibrium position if and only if the algorithm parameters are selected inside this triangle. Once the stability region is detected, the only way to find optimal parameters inside it, is to perform a wide numerical experimentation. Is important to notice that there not exist an optimal choice for these parameters that is suitable for all problems, but is important to find the best ones for each case.

4.3 Weighting inertia factor w

In this section we explain which role plays the inertia weight in updating equations (4.1.3), (4.1.4) and we show the most common updating strategies for it.

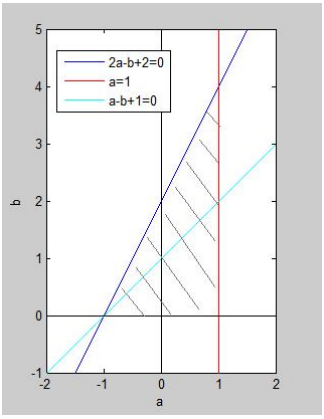


Figure 4.2.1: *Stability region.*

The inertia weight was not present in the original PSO formulation, but was introduced by Shi and Eberhart, [15] in 1998. The updating equations were:

$$v_{k+1}^i = v_k^i + c_1 r_1 (p_{best}^i - x_k^i) + c_2 r_2 (p_{best}^g - x_k^i), \tag{4.3.1}$$

$$x_{k+1}^i = x_k^i + v_k^i, \tag{4.3.2}$$

for $i = 1, \ldots, s$. Analysing (4.3.1) we can understand which is the contribution of the previous velocity to the new velocity. Without it, each particle will be "flying" toward the weighted centroid of its own best position and the global best position of the population. Under this condition, the search space statistically shrinks through the generations and PSO resembles a local search algorithm, [15]. On the other hand, by adding the first part particles have a tendency to expand the search space and so to explore new areas. So the addiction of the first part encourage a global search ability. However, both the local search and global search will benefit solving some kinds of problems, and for different problems there should be different balances between the local and the global search abilities. Considering of this, an inertia weight w has been brought into the equation (4.3.1), to play the role of balancing the global search and local search.

The inertia weight can be a constant either a positive linear or nonlinear function of k. From numerical tests it is emerged that values of w close to 1 make the PSO more like a global search method, the swarm is more capable to exploit new areas and the solution is less dependent on initial population, while for smaller values of w PSO resembles a local search method, [15].

For any optimization search algorithm it is a good idea to have a good exploration ability especially at the beginning, in order to locate the most promising areas and address the search in that directions, then near the end of the process the algorithm needs to carefully scan the local area around the selected region of the search space, in order to refine the solution approximation. Accordingly, it is convenient to define the inertia weight w as a decreasing function of the iteration index k instead of a fixed constant. A very common choice is the initialization of w to a value, w_{max}, usually close to 1, and then to adopt a linearly decreasing scheme towards zero. Usually, a strictly positive lower bound on w, w_{min}, is used to prevent the previous velocity term from vanishing. In general this linearly decreasing scheme for w is adopted, [15]:

$$w_k = w_{max} - (w_{max} - w_{min}) \frac{k}{k_{max}}; \tag{4.3.3}$$

where k_{max} is the maximum number of allowed iterations. With this choice convergence velocity is slowed down and this can be a positive feature because ensures a more accurate exploration of the search space.

4.4 The Concept of Neighbourhood

In this section we introduce the concept of neighbourhood of a particle that has been developed in order to slow down the rate of convergence of the algorithm and provide a better exploration of the search space.

Performances of the algorithm are deeply influenced by the rate of convergence. Even if fast convergence seems to be preferable, especially in applications, it could bring the swarm to collapse too quickly around a non optimal solution, so that particles can perform only local search around their convergence point. This effect of fast convergence can be mild in simple optimization problems, especially in convex problems or in problems with just one minimizer, but it becomes detrimental in high-dimensional, complex environments. So it is necessary to find the optimal rate of convergence, balancing computational costs and solution quality, [31].

There are mainly two factors that influence convergence velocity:

- the choice of the set of parameters w, c_1, c_2;
- the choice of communication scheme, i.e. the way in which particles exchange informations among each others.

In the original PSO version the following global information exchange scheme is proposed. All the particles are connected and can communicate with each other, so every particle knows instantly the overall best position at each iteration. Using this scheme, all particles assume new positions in regions related to the same overall best position, and this can reduce the exploration capabilities of the swarm.

A possible approach to overcome this problem is the introduction of the concept of neighbourhood. The main idea is the reduction of the global information exchange scheme to a local one, where information is diffused only in small parts of the swarm at each iteration. More precisely, a specific criterion of neighbourhood is established so that each particle assumes a set of other particles to be its neighbours and, at each iteration, it communicates its best position only to these particles, instead of to the whole swarm. Thus, information regarding the overall best position is initially communicated only to the neighbourhood of the best particle, and successively to all the others through their neighbours, [31].

Let $S = \{x_1 \dots x_s\}$ be the set of particles. The neighbourhood of particle $x^i \in S$ is a subset $NB^i = \{x^{i_1} \dots x^{i_r}\} \subseteq S$. For each particle x^i the best position among its neighbours is identified by calculating:

$$p_{best}^{g,i} = \arg \min_{x^j \in NB^i} f(x^j). \tag{4.4.1}$$

This is used to modify the updating equations:

$$v_{k+1}^i = wv^i k + c_1 r_1 (p_{best,k}^i - x_k^i) + c_2 r_2 (p_{best,k}^{g,i} - x_k^i); \tag{4.4.2}$$

$$x_{k+1}^i = x_k^i + v_{k+1}^i, \tag{4.4.3}$$

for $i = 1, \dots, s$, so that the particle will move towards its own best position as well as the best position of its neighbourhood, instead of the overall best position. The scheme for determining the neighbours of each particle is called neighbourhood topology.

The PSO variant that uses the overall best position of the swarm is called the global PSO variant (often denoted as *gbest* PSO), and can be considered as a special case of the variant using neighbours, that is called the local PSO variant (and denoted *lbest* PSO) to distinguish between the two approaches. In gbest PSO each neighbourhood is the whole swarm, i.e. $NB^i = S$ for all $i = 1, 2, \dots, s$. The gbest scheme is also called *star topology*, and is graphically depicted in Figure 4.4.1 on the left.

As far as the choice of a criterion of proximity is concerned, there are few possibilities, [31]. A common scheme to assign neighbours is based on equipping the search space with a proper metric. According to this, each particle would be assigned a neighbourhood consisting of a number of particles that lie closer to its current position. This approach however could be too expensive, especially when a large number of particles is used, because it requires the computation of $\frac{N(N+1)}{2}$ distances between particles at each iteration. Moreover, it exhibits a general trend of forming particle clusters that can be easily trapped in local minima. For these reasons, the idea of forming neighbourhoods based on arbitrary criteria was promoted in order to alleviate the particle clustering effect. The simplest and directly applicable alternative is the formation of neighbourhoods based on particle indices. According to this, the i-th particle assumes neighbours with neighbouring indices. Thus, the neighbourhood of x^i can be defined as:

$$NB^i = \{x^{i-r}, x^{i-r+1}, \ldots, x^{i-1}, x^i, x^{i+1}, \ldots, x^{i+r-1}, x^{i+r}\}, \tag{4.4.4}$$

where r is a parameter to be set and $|NB^i| = 2r + 1$. This scheme is illustrated in Figure 4.4.1 on the right and it is called *ring topology*.

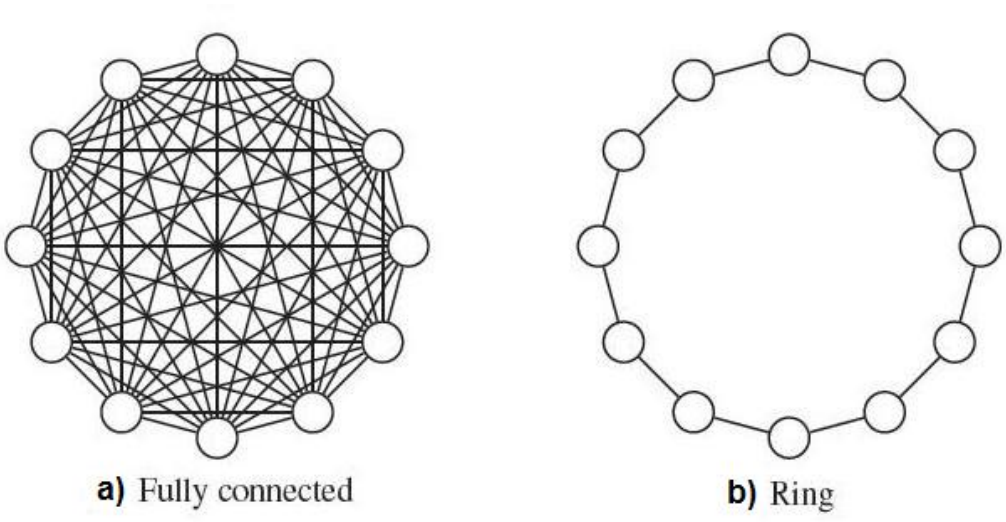


Figure 4.4.1: *Graphical representation of fully connected (a) and ring (b) topology.*

4.5 Velocity clamping

In this section we describe velocity clamping mechanism, introduced to control particles velocities.

Velocity clamping was introduced by Eberhart and Kennedy [14], in order to prevent particles from taking too large steps from their current position. More specifically, a user-defined maximum velocity threshold, $v_{max} > 0$, is considered. After determining the new velocity of each particle with equation (4.1.3), the following restrictions are applied to the velocity vector prior to the position update with equation (4.1.4):

$$(v^i_{k+1})_j = \begin{cases} v_{max} & \text{if } (v^i_k)_j > v_{max} \\ -v_{max} & \text{if } (v^i_k)_j < -v_{max} \end{cases} \tag{4.5.1}$$

where $(v^i_k)_j$ denotes the j-th component of the velocity vector of the i-th particle at k-th iteration, for $j = 1, \ldots, n$ and $i = 1, \ldots, s$. Usually different values of v_{max} are used for each component of the velocity vectors. So we have a vector of maximum velocities, $v_{max} \in \mathbb{R}^n$.

v_{max} components are usually taken as a fraction of the search space size per direction. Thus, if the search space is defined as

$$S = [(x_{min})_1, (x_{max})_1] \times \cdots \times [(x_{min})_n, (x_{max})_n], \tag{4.5.2}$$

where $(x_{max})_j$ and $(x_{min})_j$ for $j = 1, \ldots, n$ denote the j-th components of vectors x_{max} and x_{min} respectively, maximum velocity thresholds for the j-th velocity component is usually defined as, [31]:

$$(v_{max})_j = \lambda((x_{max})_j - (x_{min})_j) \tag{4.5.3}$$

for $j = 1, \ldots, n$ where $\lambda \in (0, 1]$ is a parameter to be set. Commonly the value $\lambda = 0.5$ is used, but in [25] the value $\lambda = 0.15$ is proposed and in [16] it has been empirically verified to work better.

4.6 Regrouping mechanism to prevent stagnation

In this section we describe some strategies used to prevent the swarm to be stuck during the search for the global minimum.

There can be two different situations in which the swarm does not make improvements towards the global minimum.

The first case is due to stagnation, i.e. a state in which the particles are clustered and the swarm density is really high. During the search, especially for functions to be optimized with a large number of local minima, the swarm can be trapped in a local minimizer, in this case progress toward better minima has ceased so that continued activity could only hope to refine the quality of the solution converged upon. If no particle encounters a better global best over a period of time, the swarm will continually move closer to the unchanged global best until the entire swarm has converged to one small region of the search space, and there is no improvement in the value of the objective function. Even if particles are technically always moving, no global movement of the swarm is discernible on the large scale, and the whole swarm appears as just one dot. In this case we say the swarm to stagnate. Many ideas have been proposed in the literature in order to deal with this problem, [8], [16]. Here we recall the main approaches we have tested: Random Mutation and Regrouping PSO.

Random Mutation is an occasional with a small probability random alternation of one component of p^g_{best} vector, [7]. Let $\bar{r} \sim U(1, d)$, we change the $\bar{r}$-th component of vector p^g_{best} adding, for example, 10% of the current value, [35]:

$$(p^g_{best})_{\bar{r}} = (p^g_{best})_{\bar{r}}(1 + 0.1). \tag{4.6.1}$$

Among the strategies we have tested the most effective appears to be the one proposed by George I. Evers [16]. He proposes a new PSO variant: the Regrouping PSO (PSOreg) whose goal is to detect premature convergence and free the particles from their state of stagnation to enable them to continue the search for the global minimum. When stagnation is detected particles are regrouped within a new search space large enough to escape from the minimum in which they have become trapped but small enough to provide an efficient search.

In order to detect premature convergence it is necessary to measure how near particles are to each other. Van den Bergh's Maximum Swarm Radius criterion is adopted for this purpose.

At each iteration, k, the swarm radius, $\delta(k)$ is taken to be the maximum Euclidean distance, in n-dimensional space, of any particle from the global best:

$$\delta_k = \max_{i=1\ldots s} \left(\|x^i_k - p^g_{best,k}\| \right). \tag{4.6.2}$$

Let Ω^r be the hypercube making up the search space at regrouping index r, where r is initialized to zero and incremented by one with each regrouping so that Ω^0 represents the initial search space. Let $range^{\Omega^r}$ be the vector containing the side lengths, or range per dimension, of search space Ω^r:

$$range^{\Omega^r} = \left[(range^{\Omega^r})_1, \ldots, (range^{\Omega^r})_n\right] \tag{4.6.3}$$

Let $diam(\Omega^r)$ be:

$$diam(\Omega^r) = \|range^{\Omega^r}\|. \tag{4.6.4}$$

Particles are considered too close to each other and regrouping is triggered when the normalized swarm radius, $\delta_{norm} = \frac{\delta_k}{diam(\Omega^r)}$ satisfies:

$$\delta_{norm} < \epsilon. \tag{4.6.5}$$

In [16] the value $\epsilon = 1.1 \times 10^{-4}$ is said to work well with the proposed regrouping mechanism. The dimensions of the new search space are calculated with the following formula:

$$(range^{\Omega^r})_j = \min((range^{\Omega^0})_j, \rho \max_{i\in\{1\ldots s\}} |(x_k^i)_j - (p^g_{best,k})_j|) \tag{4.6.6}$$

where $\rho = \frac{6}{5\epsilon}$, [16]. Each particle is then randomly regrouped in this new search space according to:

$$x^i_{k+1} = p^g_{best,k} + r^i range^{\Omega^r} - \frac{1}{2} range^{\Omega^r} \tag{4.6.7}$$

where $r^i \in \mathbb{R}^n$, $r^i \sim U(0,1)$, so that the new search space turns out to be

$$\Omega^r = [(x^r_{min})_1, (x^r_{max})_1] \times \cdots \times [(x^r_{min})_n, (x^r_{max})_n] \tag{4.6.8}$$

where

$$x^r_{min} = p^g_{best} - \frac{1}{2} range^{\Omega^r}; \tag{4.6.9}$$

$$x^r_{max} = p^g_{best} + \frac{1}{2} range^{\Omega^r}. \tag{4.6.10}$$

On the other hand the basic PSO approach typically converges rapidly during the initial search period and then slows, and it may happens that for many iterations, even if particles are not clustered, the swarm doesn't manage to find better positions, in terms of a lower objective function value.

Rotated Particle Swarm (RPS) is a variant of PSO algorithm introduced in [22]. In this work the authors study the performance of PSO algorithm applied to various test cases varying the problem size. It is observed that also when optimizing functions without local minima, growing the problem size the swarm is often stuck and the decrease of the objective function is really low. It is conjectured that such slow convergence of traditional PSO algorithm occurs because the velocity updating by equation (4.1.3) depends only on information of the same dimension and particles velocity is updated independently for each component of x. So it is suggested to change the update velocity scheme and to adopt a scheme that uses informations from all dimensions. Then velocity update equation (4.1.3) may be written in the following matrix form:

$$v^i_{k+1} = wv^i_k + \Phi_1(p^i_{best} - x^i_k) + \Phi_2(p^g_{best} - x^i_k) \tag{4.6.11}$$

for $i = 1, \ldots, s$, where

$$\Phi_1 = \begin{pmatrix} (c_1r_1)_1 & & \\ & \ddots & \\ & & (c_1r_1)_n \end{pmatrix}, \qquad \Phi_2 = \begin{pmatrix} (c_2r_2)_1 & & \\ & \ddots & \\ & & (c_2r_2)_n \end{pmatrix}. \tag{4.6.12}$$

Equation (4.6.11) is then replaced by

$$v_{k+1}^i = wv_k^i + A^{-1}\Phi_1 A(p_{best}^i - x_k^i) + A^{-1}\Phi_2 A(p_{best}^g - x_k^i) \tag{4.6.13}$$

where $A \in \mathbb{R}^{n\times n}$ is a rotation matrix. In this way the coordinate system is rotated and information of other dimensions is employed in calculating each component of velocity. Usually only some selected axis couples are rotated by a fixed angle θ in order to contain computational costs, so that matrix A is compound of rotation submatrices 2×2 and most of its elements are zero. In [22] is suggested to set the angle of rotation θ to $\frac{\pi}{5}$ and the number of axes to be rotated to 40% of number of dimensions.

4.7 Handling constraints

In this section we show how PSO algorithm has been recently modified in order to handle constraints.

Originally PSO algorithm, and evolutionary methods in general, were used to deal with nonlinear programming problems with non-smooth objective functions but without constraints, apart from bound constraints.

When box constraints are present:

$$x_{min} \leq x \leq x_{max} \tag{4.7.1}$$

where $x_{min}, x_{max} \in \mathbb{R}^n$, and the inequalities are component wise, the search space

$$S = [(x_{min})_1, (x_{max})_1] \times \cdots \times [(x_{min})_n, (x_{max})_n] \tag{4.7.2}$$

is bounded. However, even if particles are initialized within the search space, the application of the standard PSO update equations does not prevent particles from leaving it. On the other hand, this kind of constraints are quite easy to handle because it is simple to find feasible solutions and to repair the infeasible ones, also because it is possible to manage every component separately. The most common strategy to handle bound constraints is to bring back on the nearest boundary a particle x that has left the search space, [31]:

$$x_j = \begin{cases} (x_{min})_j & \text{if } x_j < (x_{min})_j \\ (x_{max})_j & \text{if } x_j > (x_{max})_j \end{cases} \tag{4.7.3}$$

where x_j, $(x_{min})_j$, $(x_{max})_j$ for $j = 1 \ldots n$ are the j-th component of vectors x, x_{min} and x_{max} respectively. In both cases it is also necessary to change the particle velocity, otherwise on the next iteration it is likely to have a new violation:

$$v_j = -r_3 v_j; \tag{4.7.4}$$

where v_j is the j-th component of vector v and $r_3 \sim U(0, 1)$. This strategy is depicted in Figure 4.7.1.

Only in the last few years several approaches have been proposed to extend evolutionary techniques to general NLP problems by some constraints handling method. First modifications of the Genetic Algorithms have been introduced in [1], [29] and then also Particle Swarm Optimization algorithms have been considered, [2], [10], [32]. These methods can be grouped into two main categories:

- methods based on preserving solutions feasibility;

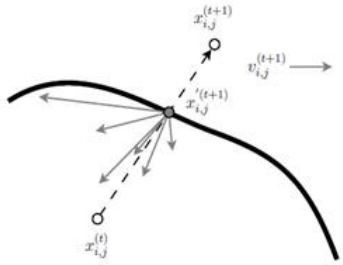


Figure 4.7.1: *Bound constraints handling strategy. Here we denote with $x_{ij}^{(t)}$ and $v_{ij}^{(t)}$ the j-th component of position and velocity vectors, respectively, for the i-th particle at the t-th iteration.*

- methods based on penalty functions.

According to the first approach, only feasible solutions are generated to maintain feasibility, and the search process is restricted within only the feasible area. This could be done by simply rejecting the infeasible individuals, but this has the obvious drawback that for problems with small feasible space it is impractical. Otherwise specialized operators were introduced to repair infeasible solutions so that a feasible solution is produced. This kind of approach is usually used with Genetic Algorithms, but the idea of rejecting individuals or modify them, is not in agreement with PSO philosophy of collaboration and re-education of bad individuals, so the second approach seems to be preferable.

The second approach is based on theory presented in Section 2.2. When using this approach, the constraints are effectively removed, and a penalty term is added to the objective function value when a violation occurs. Hence, the optimization problem is carried on minimizing the penalty function. Using this approach one has to face the difficulty of maintaining a balance between obtaining feasibility whilst finding optimality. Methods based on penalty functions were previously used with Genetic Algorithms [1], [29], in that case has been noted in [24] that too high a penalty will force the algorithm to find a feasible solution even if it is not optimal. Conversely, if the penalty is small, the emphasis on feasibility is thereby reduced, and the system may never converge.

4.8 Stopping criterion

In this section we show the standard stopping criterion used with evolutionary algorithms.

Choosing a good stopping criterion for a search algorithm is not easy because we have only informations about the value taken by the objective function and we do not have derivative informations. Indeed it is not possible to have any optimality measure, i.e. we cannot know when and if a stationary point has been reached, especially because we have no information about gradients. Then, standard stopping criteria are the followings, [31]:

- the algorithm is stopped after a maximum number of iterations or function evaluations. This criterion is usually used when one wants to find a solution in a limited amount of time. It is clear that choosing the right number of maximum iterations is critical: if that number is too low the process may be stopped when an optimal solution is not reached, if it is too high useless function evaluations are performed and the computational cost is increased.

- the algorithm stops when there are no improvement after a fixed number of iterations. Usually an improvement is measured by a sufficient decrease in the objective function or by an update of vector p_{best}^{g}. Using this criterion we introduce two more parameters to be set:

1. the number of iteration within evaluate if there are improvements,
2. the threshold that define when there is an improvement over a step.

- if an exact solution x^* of the problem is known, than the algorithm is stopped when

$$f(p^g_{best}) < |f(x^*) + \epsilon| \tag{4.8.1}$$

where ϵ is a tolerance to be fixed.

In most applications the first two criterion are used together.

4.9 A new PSO variation

We also developed a further variant of PSO algorithm with the purpose of including in the velocity update equation (4.1.3) the level of leadership of the i-th particle, and this is done modifying equation (4.1.3) adding the new term $c_3 r_3(p^g_{best} - p^i_{best})$ so that it becomes:

$$v^i_{k+1} = wv^i_k + c_1 r_1(p^i_{best,k} - x^i_k) + c_2 r_2(p^g_{best,k} - x^i_k) + c_3 r_3(p^g_{best} - p^i_{best}), \tag{4.9.1}$$

where c_3 is a weighting parameter to be set, $r_3 \sim U(0,1)$. So in the standard velocity updating equation it is added a new term that is proportional to the distance of the particle best position from the global best position. Notice that adding this term do not alter the stability analysis made in section 4.2, because the characteristic polynomial associated to the second order recursion formula obtained with equation (4.9.1) is the same of (4.2.7). The addition of the new term indeed modify only the constant term.

Numerical tests have shown that adding this term at each iteration does not provide good results, but adding it only when the swarm seems to be stuck, i.e. when after two following iterations a better position is not found, helps the swarm to escape from this stalemate, improving also the rate of convergence.

In the numerical results section, we will address to this version of PSO algorithm with the term *PSO*c_3.

4.10 Our PSO implementation in ECOS

In this section we describe the PSO algorithm implemented in ECOS. We have tested various features of PSO method on different examples of energy districts and we have chosen the version of the algorithm that is most suitable for solving problem (1.2.1). We had to choose the swarm size, the free coefficients, the topology scheme, a constraint handling strategy, a stopping criterion. It is important to underline that optimizing this kind of problem is really different from optimizing a test function, even a particularly difficult one. First of all we have to deal with problems with hundreds of variables, with an objective function that has not a simple analytic form and that is expensive to evaluate, and we have also to deal with nonlinear constraints. Moreover the algorithm must return results quickly to be useful in practise. All these considerations have deeply influenced our choices.

Regarding the number of particles, it is impossible to use wide swarms because each particle of the swarm requires two function evaluations, the objective and the constraints functions, at each iteration. We underline that the evaluation of those functions in this case is very expensive, as opposed to problems in which a simple test function is optimized. Time required for every function evaluation is quite long, so using a high number of particle becomes prohibitive. On the other hand using few particles means low exploration capability. After several numerical

tests, we have chosen to use 20 particles, as a good compromise between solution quality and execution time.

After a wide numerical experimentation we set $c_1 = 1.3$ and $c_2 = 2.8$, we adopted the linearly decreasing scheme (4.3.3) for choosing w, with $w_0 = 0.6, w_{max} = 0.1$.

Regarding the choice of topology we have preferred the global PSO version that guarantees a faster convergence, in order to have shorter execution times.

Regarding the constraints handle strategy, for bound constraints we adopted the approach introduced in Section 4.7, while for nonlinear constraints we have tested many penalty function approaches, [35], but the most adequate turned out to be the one described in [1] and [29]. We tested its performance both on some analytical problems and on problem (1.2.1), [35]. Such approach is called *annealing penalties*, and it is based on the quadratic penalty function (2.2.6). For this approach the penalty term $\tau = \frac{1}{\nu}$ is used and it is called temperature. Here we sketch the k-th iteration of such a procedure:

1. Given τ_k.
2. Evolve the population using PSO algorithm with the following penalty function:
$$\Phi(x, \tau_k) = f(x) + \frac{1}{2\tau_k} \sum_{i \in \mathcal{I}} \max(0, g_i(x))^2. \tag{4.10.1}$$
3. Decrease temperature τ_k: choose $\tau_{k+1} < \tau_k$.

After several numerical tests we decided to change the penalty parameter τ_k at each iteration, instead of solving more subproblems with fixed decreasing penalty parameters, as it is done in [1] and shown in the above scheme. With this choice the swarm appears more free to explore the search space, while in the other case it was difficult to have a good balance between obtaining a good solution and a feasible solution, the swarm was more likely stuck in local feasible point and the obtained value of the objective function was too high.

Regarding the choice of a mechanism to prevent the swarm to be stuck in local minima, we have tested all the strategies described in Section 4.6 but no one of them seemed to improve PSO performance significantly. Among them we decided to adopt the regrouping mechanism even if its contribution is not so considerable.

As far as the velocity clamping is concerned, we have tested its effect on the algorithm performance using different values of λ, but in this case it does not seem to be effective, so it was not used.

Regarding the stopping criterion we adopted both the approaches described in Section 4.8, i.e we stop PSO algorithm when a maximum number of iterations k_{max} is performed or when there are no improvements over a fixed number of iterations κ. We measure an improvement in terms of a decrease of the objective function, and we judge that the decrease is not sufficient when the following condition is satisfied for κ consecutive iterations:

$$\frac{|f(x_{k-2}) - f(x_k)|}{|f(x_{k-2})|} < \epsilon \tag{4.10.2}$$

where ϵ is a tolerance to be fixed.

On problem 1.2.1 we have tested also the new version of PSO algorithm we have introduced in Section 4.9. As it will be shown in Chapter 5, from numerical tests it has emerged that PSOc_3 has a good performance especially at the beginning of the optimization process. The addition of the new term in the velocity updating equation indeed improves the convergence rate and provides a quicker decrease of the objective function. Nevertheless the swarm is easier trapped in local minima, so it is not advisable to use this scheme at the end of the optimization process. Considering of this, we decided to use an **hybrid PSO version**, i.e. to use PSOc_3 in the first half of iterations and basic PSO in the last ones.

Chapter 5

Numerical tests and results

In this chapter we show the results of the application of the PSO and SLP algorithms we have developed to some test cases, and we evaluate their performance. First we have optimized the Rastrigin function, [33], a famous benchmark problem widely used in the literature to test the performance of Evolutionary Algorithms. The numerical experimentation has been carried out on a PC equipped with a Intel(R) Core(TM)2 DUO CPU P8400 @ 2.26 GHz, 4.00 GB RAM, Windows Vista SP2 32 bit and using Matlab R2012a. Then we have dealt with a real life problem, the optimization of energy districts, that we have introduced in Chapter 1. The numerical experimentation has been carried out on a PC equipped with a Intel(R) Core(TM) i5-2400 CPU 3.10 GHz, 4.00 GB RAM, Windows 7 Professional 32 bit and using Matlab R2012b.

For all the test cases we report results that are the average of those obtained over 10 runs. In fact, PSO is a stochastic algorithm and each run yields a different solution. Moreover also the starting swarm is different because particles are initialized randomly. As far as the results obtained with SLP algorithm are concerned, since it is a deterministic algorithm, we report the average of the results obtained over ten runs, where in each run the starting point is changed randomly. For the optimization of energy district the starting point has been chosen such that all the constraints (1.2.1b), (1.2.1c) hold, otherwise the SLP approach described in Section 1.3, to which we will refer below with the name SLP_1, may not converge. On each run the starting point is the same for SLP_1 and for the SLP approach based on penalty function theory, to which we will refer below with the name SLP_2, so that we can compare their performance regardless of the initial guess. The results are reported in several tables in which the headings of the columns have the following meaning:

- **solver** is the name of the solver used for the optimization process;
- $\bar{\boldsymbol{f}}$ is the arithmetic mean of the values f^i, for $i = 1, \ldots, 10$, where f^i is the objective function value obtained at the i-th run: $\bar{f} = \frac{1}{10} \sum_{i=1}^{10} f^i$;
- $\boldsymbol{\sigma_f}$ is the standard deviation of values f^i: $\sigma_f = \sqrt{\frac{\sum_{i=1}^{10} (f^i - \bar{f})^2}{10}}$;
- $\mathbf{max}\ \boldsymbol{f}$ is the maximum among the values of the objective function obtained over the 10 runs: $\max f = \max_{i \in \{1,\ldots,10\}} f^i$;
- $\mathbf{min}\ \boldsymbol{f}$ is the minimum among the values of the objective function obtained over the 10 runs: $\min f = \min_{i \in \{1,\ldots,10\}} f^i$;
- $\bar{\boldsymbol{k}}$ is the arithmetic mean of the values k^i for $i = 1, \ldots, 10$, where k^i is the number of iteration required by the i-th run: $\bar{k} = \frac{1}{10} \sum_{i=1}^{10} k^i$;

- $\boldsymbol{\sigma_k}$ is the standard deviation of values k^i: $\sigma_k = \sqrt{\frac{\sum_{i=1}^{10}(k^i - \bar{k})^2}{10}}$;
- $\mathbf{max}\, \boldsymbol{k}$ is the maximum among the number of iterations required by the optimization process over the 10 runs: $\max k = \max_{i \in \{1,\dots,10\}} k^i$;
- $\mathbf{min}\, \boldsymbol{k}$ is the minimum among the number of iterations required by the optimization process over the 10 runs: $\min k = \min_{i \in \{1,\dots,10\}} k^i$;
- $\|\boldsymbol{x}\|_{\mathbf{2}}$ is the 2-norm of the solution, obtained as an average of those of the 10 solutions found. This column is present only in the tables in Section 5.1 because when optimizing the Rastrigin function this parameter is useful to have a measure of the distance of the solution found from the global minimum that is $x = 0$;
- **time/iter(·)** is the time required for an iteration of the optimization process, the value in brackets is the unit of measurement, namely s are seconds and m are minutes;
- **time(·)** is the total time required for the optimization process, the value in brackets is the unit of measurement, namely s is seconds and m is minutes.

As far as the stopping criterion is concerned, all of the following results are obtained:

- stopping the PSO algorithm when the following condition:

$$\frac{|f(x_{k-2}) - f(x_k)|}{|f(x_{k-2})|} < 10^{-3} \tag{5.0.1}$$

  is satisfied for 100 consecutive iterations for the Rastrigin function, for 40 consecutive iterations for the optimization of energy district. For the Rastrigin function we allowed 100 iterations instead of 40 because the problem has a very high number of local minima and it has been observed that it is more likely for the swarm to be stuck in one of them so that the objective function doesn't decrease for some iterations but then decreases again;

- stopping SLP_1 when the following condition is satisfied:

$$\|d_k\| < 10^{-8} \tag{5.0.2}$$

  where d_k is the current step;

- stopping SLP_2 when the following conditions are satisfied:

$$\max\{\|\nabla f(x_k) + J(x_k)^T \lambda_k\|_\infty, \|g(x_k)^T \lambda_k\|_\infty\} < \epsilon(1 + \|\lambda_k\|_2); \tag{5.0.3}$$

$$\max\{\max_{i \in \mathcal{I}}(0, g_i(x_k)), \max(0, x_{min} - x_k), \max(0, x_k - x_{max})\} < \epsilon(1 + \|x_k\|_2); \tag{5.0.4}$$

  where x_k and λ_k are current solution and multipliers vector approximations respectively and where we have set $\epsilon = 10^{-3}$.

Besides these stopping criterion, the execution is stopped whether a maximum number of iterations, k_{max} is reached. For evolutionary methods, and especially in the last iterations, the objective function decreases really slowly and sometimes during an iteration it does not decrease at all. So the maximum number of iterations should be reasonably high. Case to case one can chose the most appropriate value, considering the available time and the desired solution quality.

For the 2-dimensional Rastrigin function we have set $k_{max} = 500$ for PSO algorithm and $k_{max} = 50$ for SLP algorithm; for the 30-dimensional Rastrigin function we have set $k_{max} = 1200$ for PSO algorithm and $k_{max} = 50$ for SLP algorithm; for the problem of energy district, for PSO algorithm we have set $k_{max} = 1000$ for the first phase, cf. Section 1.4, and $k_{max} = 500$ for the second phase, for SLP algorithm we have set $k_{max} = 30$ for the first phase and $k_{max} = 70$ for the second phase.

Notice that SLP_2 applied to the Rastrigin function test case reduces to SLP_1 because there are not nonlinear constraints, the only difference lays in the stopping criterion. At the same time in PSO method the penalty function (4.10.1) reduces to the objective function f.

5.1 Early PSO application: the Rastrigin function

In this section we show the results of the optimization of the Rastrigin function by SLP and PSO algorithms. We have mainly focused to test various features of PSO algorithm, such as swarm size, neighbourhood topology, various choices of coefficients.

The Rastrigin function, see Figures 5.1.1, 5.1.2, is a typical example of nonlinear, nonconvex function, usually used as a performance test problem for optimization algorithms, especially for search algorithms. It was first proposed by Rastrigin as a 2-dimensional function [33] and has been generalized by Mühlenbein [3] to larger dimensions. Finding the global minimum of this function is a fairly difficult problem, due to its large search space and its large number of local minima. It is defined by:

$$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]; \tag{5.1.1}$$

where n is problem dimension and $x \in [-5.12, 5.12]^n$. It has a global minimum at $x = 0$ where $f(x) = 0$.

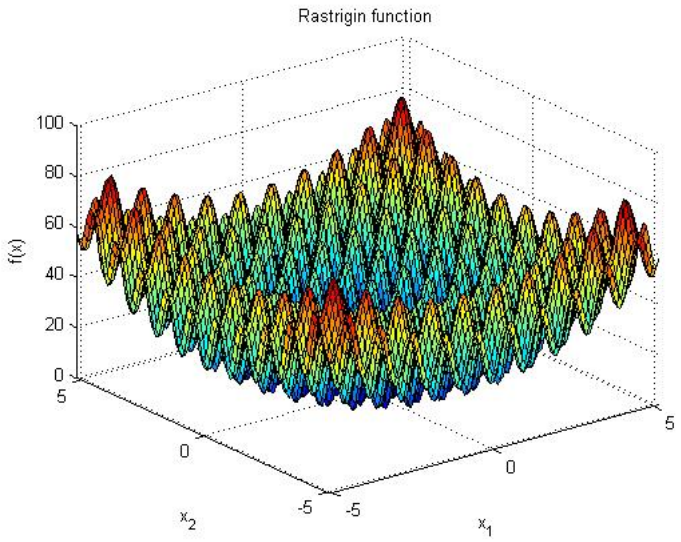


Figure 5.1.1: *Rastrigin function.*

Among optimization problems, some are regarded as more difficult than others, for example problems with many local minima are considered more difficult than others. Clerc in [9] gives the following formal definition: difficulty of an optimization problem in a given search space is the probability of not finding a solution by choosing a random position according to a uniform distribution. It is thus the probability of failure at the first attempt.

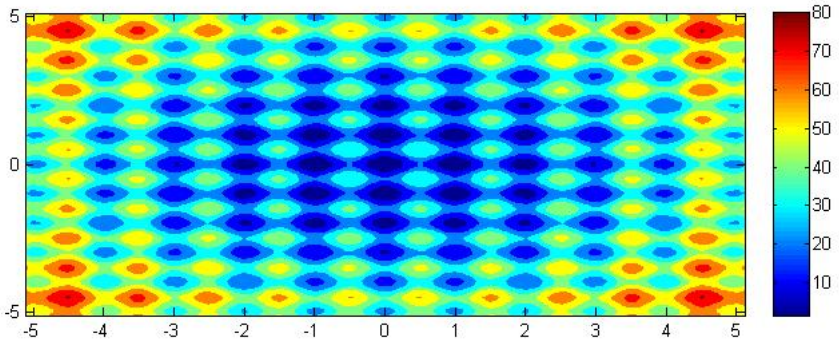


Figure 5.1.2: *Contour plots of the Rastrigin function.*

When high precision to find a global minimum is required, the probability of failure is very high and to take it directly as a measure of difficulty is not very practical. Thus Clerc, to practically compute the difficulty of a problem, suggests to the following formula:

$$difficulty = -\log(1 - failure\ probability) = -\log(success\ probability) \tag{5.1.2}$$

The success probability can be estimated in various ways, according to the form of the objective function, [9], by direct calculation in simple analytical cases or it can be numerically approximated, through Montecarlo approaches.

Minimizing Rastrigin function is an optimization problem of the form:

$$\min_x f(x) \tag{5.1.3a}$$

subject to

$$-5.12 \leq x_i \leq 5.12, \qquad i = 1, \ldots, n. \tag{5.1.3b}$$

Optimizing this function becomes more and more difficult as the problem size grows.

We have solved (5.1.3) using both PSO and SLP starting with the two dimensional case, the easiest one, which is particularly interesting because allows us to graphically represent the evolution of the swarm during its search of the global minimum. Then we have set $n = 30$ to test the effect of problem size to the optimization process.

5.1.1 2-dimensional Rastrigin function

In this section we compare the results obtained by PSO algorithm using 10 particles, a star topology and the following standard set of parameters typically used in the literature when optimizing the Rastrigin function or other famous benchmark functions, [14], [16]:

- $c_1 = c_2 = 1.4961$;
- $w = 0.72$;

and those obtained by the two versions of SLP algorithm, varying randomly the starting solution.

As we can see from Table 5.1.1, while for PSO algorithm the Rastrigin problem is a fairly easy problem to solve and the algorithm never misses the goal, regardless of the initial positions of particles, the SLP algorithm is not so reliable. On 10 runs SLP never reaches the goal, but it stops on local solution far enough from the global minimum. We can also notice that even if PSO performs more iterations the execution time is really low. The results obtained

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	$\|x\|_2$	time(s)
PSO	10^{-7}	3×10^{-7}	10^{-6}	10^{-13}	273	30	332	220	10^{-5}	2
SLP_1	26.1	11.7	49.7	3.9	31	4	38	27	4.9	1
SLP_2	25.1	12.2	38.9	4.9	3	1	5	3	4.6	0.2

Table 5.1.1: *Comparison between SLP and PSO on 2-dimensional Rastrigin function.*

by SLP algorithm are strongly dependent on the starting point: the final solution is the local minimum nearest to the starting point, as it is expected by a method designed to converge to a local minimum. On the other hand from these results PSO algorithm capability to converge to the global minimum appears clearly. These considerations are evident from Figures 5.1.3 and 5.1.4. In Figure 5.1.3 are depicted the contour levels of Rastrigin function and in blue the 10 particles of the swarm. The four figures refer to different stages of evolution and to a single run: in them are depicted the position of particles at the beginning of the optimization process, after 50 and 100 iteration and at the end of the process. We can see that at the beginning particles are uniformly distributed within the search space, then they begin to approach zero until they converge to it. On the other hand, in Figure 5.1.4 we show in different colours ten different runs of SLP_1 algorithm. In the four figures we can see the evolution of those solutions to the varying of iterations, in the first are depicted the initial solutions and in the others the solutions after 5 and 10 iterations and at the end of the optimization process, approximately after 35 iterations. In this case it is evident the difference between the local convergence of SLP and the global one of PSO.

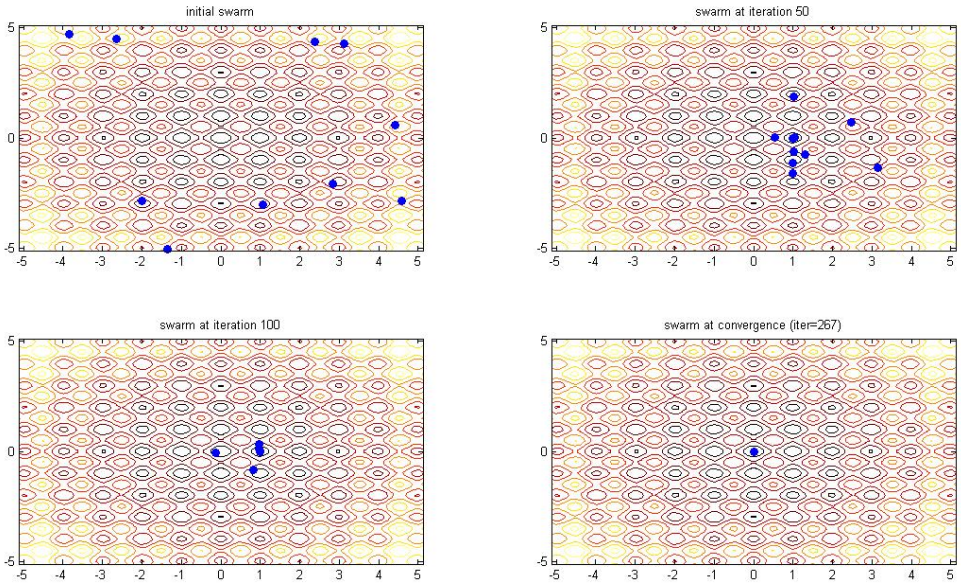


Figure 5.1.3: *Evolution of the swarm during the optimization process, PSO solver, star topology.*

We can also compare the evolution of the swarm gained using both a star topology and a ring topology, which is depicted in Figure 5.1.5. We can see that using this second topology the swarm tends to cover a larger part of the search space and to converge later.

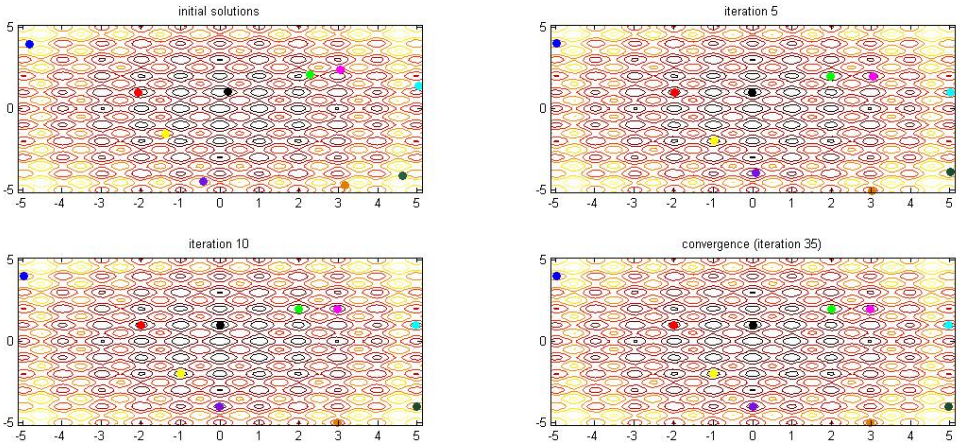


Figure 5.1.4: *Evolution of ten different solution, SLP solver.*

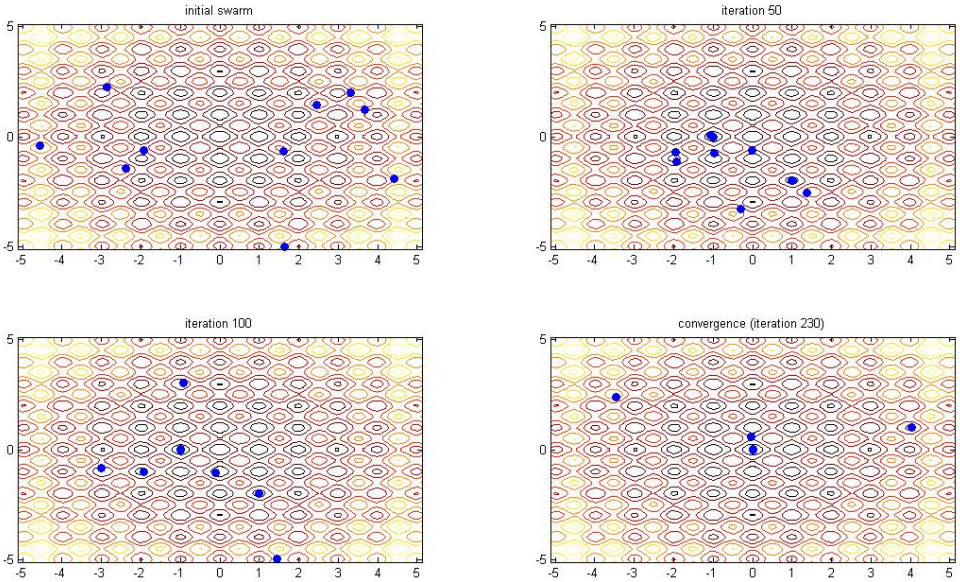


Figure 5.1.5: *Evolution of the swarm during the optimization process, PSO solver, ring topology.*

Swarm size	$\bar{f}$	σ_f	$\max f$	$\min f$	$\bar{k}$	σ_k	$\max k$	$\min k$	$\|x\|_2$	time(s)
30	63.8	15.2	99.4	39.7	752	98	926	604	8.2	4
60	60.3	14.2	79.5	27.8	676	62	809	604	7.7	7
120	43.0	11.2	68.6	16.9	645	63	791	597	6.3	14

Table 5.1.2: *PSO with different swarm sizes.*

5.1.2 30-dimensional Rastrigin function

In this section we have used this more difficult benchmark problem to test many different features of the PSO algorithm, such as:

- different choices of swarm size;
- different choices of the inertia weight; namely we have compared PSO with a fixed value of w with PSO using w linearly decreasing with iterations;
- different choices of neighbourhood; namely we have compared PSO with star and ring topology;
- velocity clamping;
- regrouping mechanism.

All the results reported in the following sections are obtained with the standard set of parameters $c_1 = c_2 = 1.4961$. In the following sections we will show the obtained results, all of them have been obtained repeating 10 runs of the algorithm. It is important to notice that compared with the 2-dimensional problem, this problem is really more difficult to solve and even PSO algorithm does not manage to find the global minimum.

Number of particles

In order to test the effect of different swarm sizes on the algorithm's performance, we have chosen $w = 0.72$, a star topology, and we have compared the results obtained with three different swarm sizes: 30, 60, 120.

We can see from Table 5.1.2 that increasing the swarm size ensures a better exploration of the search space providing a lower mean value for the objective function and also a smaller standard deviation. On the other hand it is important to notice that increasing the swarm size imply more function evaluations (about 22500 for $s = 30$, 40500 for $s = 60$ and 77400 for $s = 120$) and so higher computational costs and execution times. So, in choosing the swarm size, it is crucial to make a good compromise, balancing computational costs and solution approximation quality, in connection with the available time. Notice that also growing the swarm size the global minimum isn't reached.

Choice of the inertia weight and of the Neighbourhood topology

In this and in the following sections, according to the above remarks, for the numerical tests we have chosen swarm size $s = 60$. In this section we show the numerical results of the application of PSO algorithm with two different choices for the inertia weight w:

- w linearly decreasing with iterations;

w	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(s)
variable	52.8	10.2	77.1	34.8	892	66	1011	767	8.4
fixed	60.3	14.2	79.5	27.8	676	62	809	604	6.5

Table 5.1.3: *Comparisons between fixed w and linearly decreasing scheme, fully connected topology.*

- w set to a constant value;

and with two different choices of neighbourhood topology:

- the fully connected topology;
- the ring topology.

As far as the inertia weight w is concerned, when the linearly decreasing scheme is adopted we set $w_{max} = 0.9, w_{min} = 0.1$ and $w_k = w_{max} - (w_{max} - w_{min})\frac{k}{k_{max}}$, while the fixed value of w used is $w = 0.72$. On the other hand, regarding the neighbourhood topology, for the ring topology we have chosen the size of neighbourhood, referring to Section 4.4, to be $r = 2$. The results in Table 4.3.3 are obtained with a star topology. In the first line of Table 4.3.3, w linearly decreasing is used, in the second line $w = 0.72$. We can see that using w linearly decreasing ensures better performance, gaining a reduction of the average value of the objective function of 15% with respect to the one obtained using a fixed value, and providing a smaller standard deviation. On the other hand this choice involves higher computational costs because it requires more iterations to converge, so more function evaluations and higher execution times. Plotting the values of the objective function obtained during the optimization process it is possible to understand the reason behind these results. As we can see from Figure 5.1.6 on the right, the choice of a fixed value of w provide a very quickly decrease especially in the firsts 200 iterations, although it is more likely for the solution to be trapped in a local minimum because the convergence rate is too high. On the left we compare the decrease of the objective function gained adopting the linearly decreasing scheme. The objective function decreases really slower, after 200 iterations its value is still very high while in the other case it was close to the final value, but at the end a lower value is reached.

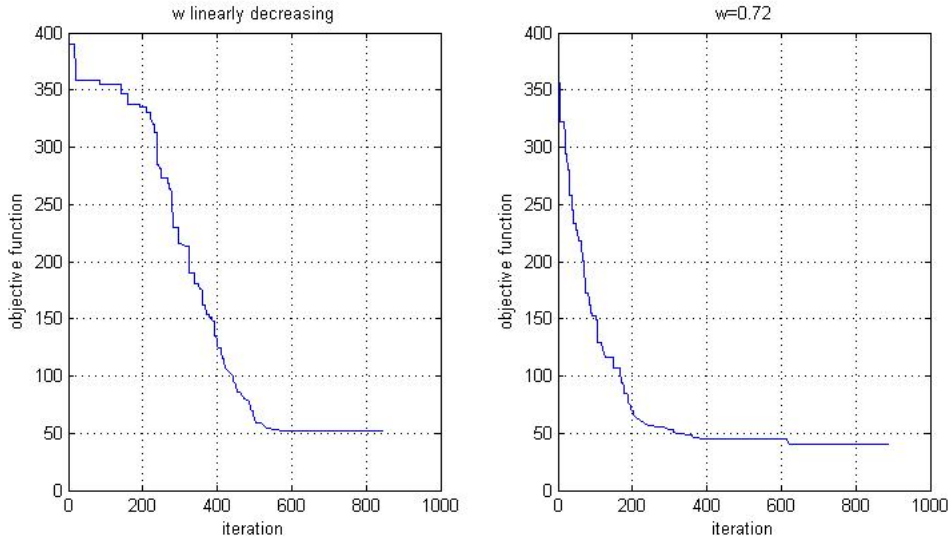


Figure 5.1.6: *Decrease of the objective function provided by two different choices of w, star topology.*

It is also important to notice that the value of fixed w we have used was recommended for this specific problem and so it is the best choice for this case. Using different values one could

w	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(s)
variable	66.1	12.2	85.6	45.8	859	47	1000	834	10
fixed	56.4	14.2	88.8	42.80	841	10	860	826	9

Table 5.1.4: *Comparisons between two different choices of w for PSO with ring topology.*

obtain worst results, and also dealing with another problem this could not be the best choice to perform. Also for this reason it should be preferable to choose a linearly decreasing strategy, because adopting a fixed value for w requires a starting phase for detecting the best value for the specific problem.

Results in Table 5.1.4 are obtained adopting a ring topology and both using w linearly decreasing with iterations in the first line and $w = 0.72$ in the second line.

Comparing results of Tables 5.1.3 and 5.1.4 we can notice that:

- when a fixed value of w is used, it is preferable to choose a local version of PSO algorithm: using a less connected topology slows down convergence and provide better performance. In Figure 5.1.7 we can compare the decrease of the objective function in the two cases, notice that using the ring topology the decrease is slower.
- using a linearly decreasing strategy for w a global topology seems to provide better results.

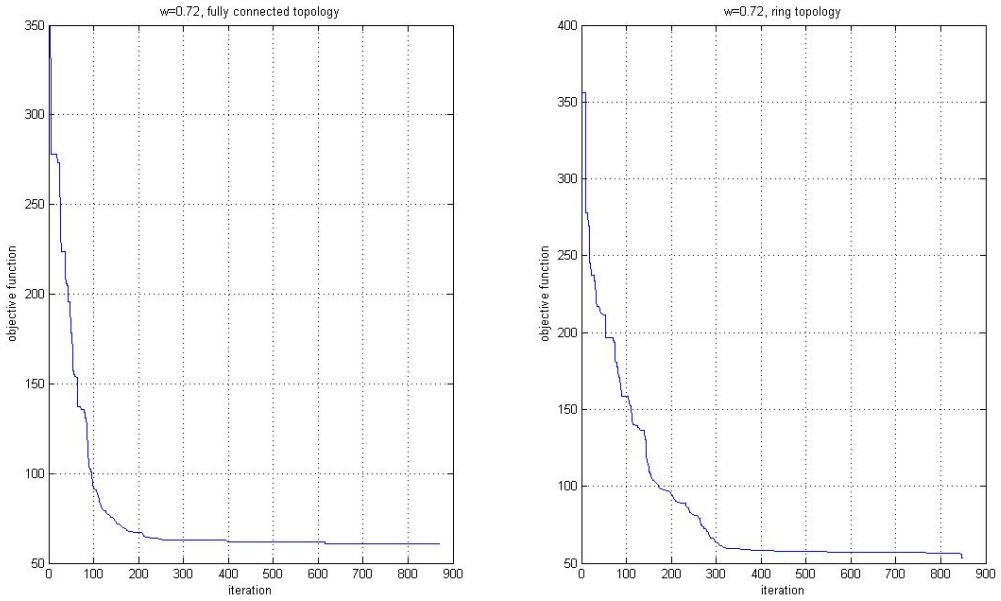


Figure 5.1.7: *Comparison between star and ring topology with $w = 0.72$.*

Velocity clamping

In this section we compare basic PSO with PSO with velocity clamping described in Section 4.5. We chose, as suggested in [16], $\lambda = 0.15$. Comparing Table 5.1.5 with Table 5.1.2 we can see that using velocity clamping actually improves the performance of PSO algorithm, providing solutions nearer to the global minimum, lower values of the objective function and of the standard deviation.

swarm size	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(s)
30	45.7	9.6	59.6	30.8	749	88	940	656	6.5
60	39.5	9.3	56.7	28.8	780	80	930	650	6.1
120	32.3	7.9	48.7	17.9	782	92	922	667	5.6

Table 5.1.5: *PSO with various swarm size and velocity clamping.*

algorithm	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	$\|x\|$
basic PSO	32.6	6.6	40.8	20.9	780	80	930	650	6.1
PSOreg	20.4	4.8	28.8	13.9	678	54	960	639	4.8

Table 5.1.6: *Comparison between basic PSO and PSO with regrouping mechanism.*

PSO with regrouping mechanism

We have seen that growing the problem dimension minimizing the Rastrigin function is a fairly difficult problem and that the swarm tends to be trapped in one of the many local minima. To deal with this problem we have tested all the strategies described in Section 4.6. Rotated Particle Swarm, as it is shown also in [22], doesn't improve PSO performance on the 30-dimensional Rastrigin function, but in [22] it is shown to be effective on higher dimensions (n=400). Also Random Mutation does not seem to improve PSO performance, while the regrouping mechanism proposed by George I.Evers [16] seems to be really effective. In Table 5.1.6 are compared the results obtained by basic PSO algorithm and the one using the regrouping mechanism. These results have been obtained setting $w = 0.72$ and choosing a star topology.

We can observe that using PSO with regrouping mechanism, denoted by PSOreg, we obtain lower values of the objective function on a lower number of iterations.

In [16] it is suggested to use a fixed value of w in conjunction with the regrouping mechanism, because it has been found to be more beneficial to allow the quick convergence of the static weight and to regroup at premature convergence than to take a considerably longer time to converge cautiously and to regroup less often. PSO algorithm is stopped with the stopping criterion (5.0.1) reported at the beginning of this chapter, but while this criterion is suitable for the basic version of PSO algorithm, to appreciate completely the benefits of the application of such regrouping mechanism, a wider number of iteration is necessary. In fact, also providing a higher value for k_{max} when using basic PSO, if the swarm is stuck in a local minimum it doesn't manage to escape from it. On the other hand when using PSOreg allowing an high number of iterations the swarm manages to make little progress towards the global solution until it reaches a local minimum near to it. A stopping criterion that fits better with this version of PSO algorithm is the following: PSO algorithm is stopped when a maximum number of function evaluations is performed. Note that the required number of iteration to obtain good results (i.e. a value of the objective function less than 2) is really high: using 20 particles the maximum number of function evaluations we used was 1000000, it means 50000 iterations and an execution time of 3.5 minutes. In Figure 5.1.8 is depicted the decrease of the objective function during the optimization process: notice that also if for several iterations no better positions are found, then little progress is made and as the swarm approaches the global minimum, a progressively lower value of the objective function is obtained.

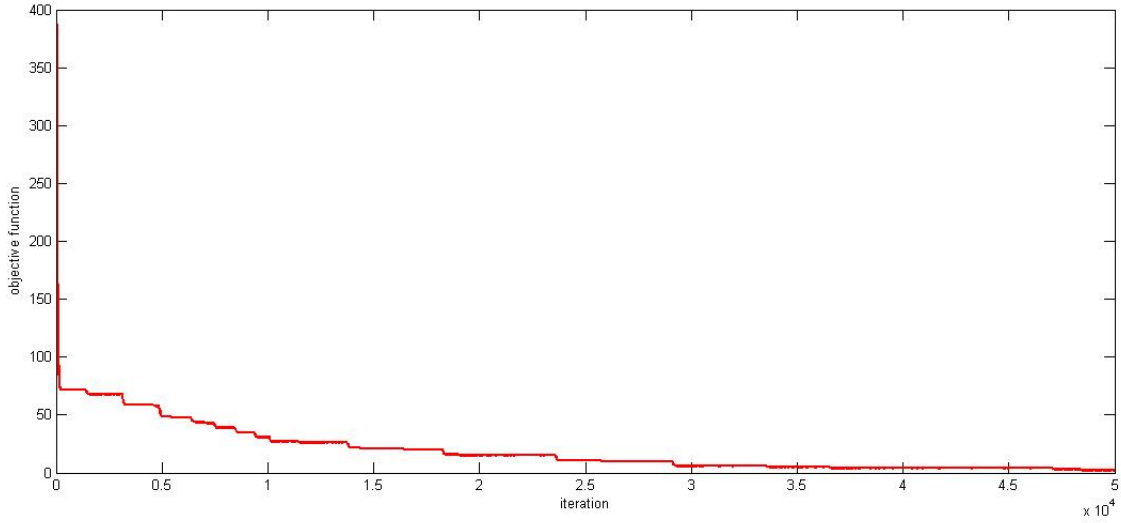


Figure 5.1.8: *Decrease of the objective function when using PSO reg allowing a wider number of function evaluations.*

solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	$\|x\|$	**time(s)**
basic PSO	41.0	5.7	49.7	33.8	551	42	612	482	6.9	5.2
PSOc_3	49.5	9.7	63.6	35.8	460	21	501	429	7.1	4.3
hybrid PSO	40.4	8.4	51.7	27.8	553	26	615	521	6.6	5.3

Table 5.1.7: *Comparisons between basic PSO and new variation PSO c_3.*

PSOc_3

In this section we show the results obtained with the new variation of PSO algorithm, PSOc_3, we introduced in Section 4.9. As we have disclosed in Section 4.10, from several numerical tests it has emerged that PSOc_3 shows a good performance at the beginning of the iterative process, gaining a good convergence rate and a quicker decrease of the objective function, but is easier trapped in local minima. This behaviour emerges clearly comparing the first two lines of Table 5.1.7, in which are compared the performance of basic PSO and PSOc_3 both with w linearly decreasing, star topology, velocity clamping, swarm size 60. In Figure 5.1.9 is depicted the decrease of the objective function provided by both basic PSO and PSOc_3, from which it is evident that the two algorithms have different rates of convergence. As we have disclosed in Section 4.10, in order to overcome this problem and to get benefits from both the approaches we tested also an *hybrid PSO algorithm*, i.e. we used PSOc_3 in the first half of iterations and basic PSO in the last ones, specifying for each algorithm a maximum number of iteration $k_{max} = 500$. In the last line of Table 5.1.7 are reported the result of combination of the two approaches: the algorithm has a good performance and do not stall too early. Regarding the value of the free parameter c_3, the value $c_3 = 1$ has been used in the numerical experimentation and it proves to work well.

It is worth noticing that, like in the 2-dimensional case, the time required for the optimization process by all these versions of PSO algorithm is really low, even if PSO algorithm requires an high number of iterations to converge.

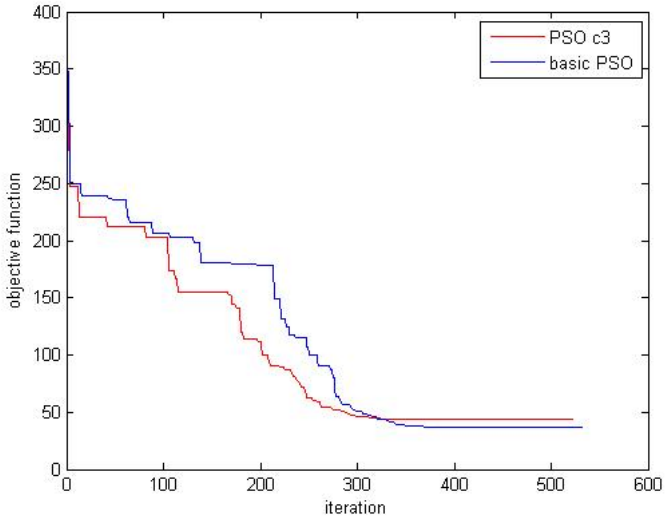


Figure 5.1.9: *Decrease of the objective function provided by PSOc_3 and basic PSO.*

solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	$\|x\|$	time(s)
hybrid PSO	40.4	9.7	51.7	27.8	765	67	918	652	6.6	7
SLP_1	243.6	50.4	342.9	163.7	40	2	43	38	15.4	2
SLP_2	276.4	37.8	369.8	210.4	5	2	11	3	15.7	0.5

Table 5.1.8: *Comparisons between SLP and PSO on 30-dimensional Rastrigin function.*

SLP application to 30-dimensional Rastrigin function

If in dimension 2 SLP doesn't give excellent results dealing with this problem, in higher dimension the situation is even worst, as it is shown in table Table 5.1.8. SLP indeed in all runs provides a really high value of the objective function and a solution far from the global minimum. So, when optimizing functions with a lot of local minima, the capacity of the evolutionary algorithm to search for the global minimum appears clearly and the use of such an algorithm provides better results.

5.2 Optimization of energy districts

In this section we describe the PSO variation we used for the optimization of energy districts and the results of the numerical tests made on 13 different realistic models of energy districts, obtained with PSO and the two versions of SLP algorithm.

The PSO Matlab code we developed was made available in the software tool previously implemented by Enel Ingegneria e Ricerca and provided as an alternative to the SLP solver already implemented in the code. The software tool has been equipped with Matlab functions building up the objective, the constraints functions and evaluating gradients of the objective and of the constraints functions. When it is possible the analytical expression of gradients is used, otherwise they are approximated by finite differences. This is a quite heavy calculation and it is unnecessary when using PSO algorithm. So we needed to modify the existing codes in order to avoid this calculation when the new solver is used.

The version of PSO algorithm we used is described in Section 4.10. We recall some of its features:

- we used an *hybrid version* of PSO algorithm, consisting of PSOc_3 for the first half of iterations, with $c_3 = 1$, and basic PSO for the last ones;
- the swarm size is 20;
- $c_1 = 1.3$ and $c_2 = 2.8$;
- for w a linearly decreasing scheme is adopted, $w_k = w_{max} - (w_{max} - w_{min})\frac{k}{k_{max}}$, with $w_{max} = 0.6$, $w_{min} = 0.1$ and $k = 1, \ldots, k_{max}$;
- the topology scheme adopted is a star topology;
- the stopping criterion is the one described in Equation (5.0.1) with a fixed tolerance $\epsilon = 10^{-3}$;
- the constraints handling strategy for bound constraints is the one described in Section 4.7, for nonlinear constraints is the one described in Section 4.10, with $\tau_0 = 0.1$ and $\tau_{k+1} = (1 - 0.01)\tau_k$;
- the regrouping mechanism described in Section 4.6 is adopted.

We have tested all the strategies presented in Section 4.6 to deal with the problem of stagnation, but no one of them appears to be really effective dealing with problem 1.2.1. In Figure 5.2.1 we can compare the decrease of the objective function provided by basic PSO and RPS, both of them with the basic scheme described above, when applied to an example of energy district, and we can notice that the two algorithms have almost the same performance. Also Random Mutation did not provide good results. Then, we decided to adopt the regrouping mechanism even if, while it improves significantly PSO performance on the Rastrigin function, in this case it is rarely applied within the maximum number of iterations specified and its contribution is not so significant. In fact this strategy would require a large number of iterations in order to be effective, as we have noticed in Section 5.1.2. On the other hand, due to the long time required by the optimization process, it is not possible to allow an higher number of iterations.

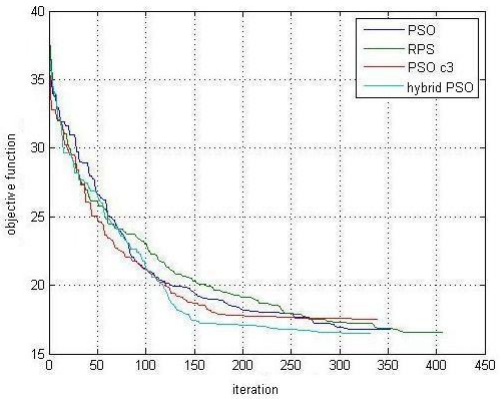


Figure 5.2.1: *Decrease of the objective function provided by different versions of PSO algorithm.*

From Figure 5.2.1 we can also analyse the behaviour of the new version of PSO algorithm we introduced in Section 4.9, PSOc_3, compared to the one of basic PSO algorithm. As we have already noticed in the previous section, it has a good performance at the beginning gaining a good convergence rate and a quicker decrease of the objective function, but is easier trapped in local minima. Again in Figure 5.2.1 is possible to see the result of the combination of the two approaches, i.e. the behaviour of the hybrid PSO version we have introduced in Section 4.10: the algorithm has a good convergence rate and does not stall too early. This is the version of

Solver	$\bar{f}$	σ_f	$\max f$	$\min f$	$\bar{k}$	σ_k	$\max k$	$\min k$	time/iter(s)	time(m)
PSO	16.0	0.2	16.4	15.7	1095	54	1193	1019	0.08	1.5
SLP_1	15.9	0.5	17.4	15.7	26	13	41	5	0.1	0.02
SLP_2	15.7	0.1	15.9	15.5	9	2	12	5	0.07	0.01

Table 5.2.1: *'Test1 Elettrico', comparison of solvers.*

PSO algorithm we have decided to use for the optimization of energy district, then from now on with the generic term PSO we will refer to this hybrid version. As for the Rastrigin function, the value $c_3 = 1$ has been used in the numerical experimentation and it proves to work well.

The three algorithms have been tested to 13 different realistic models of energy districts, in the following sections we describe each test case and the results of the optimization, comparing the performances of the algorithms. For each test case we show a table with the results of the optimization provided by PSO algorithm, and both versions of SLP algorithm, SLP_1 and SLP_2. We postpone in Section 5.2.14 some remarks about the results obtained and some comparisons among the three algorithms.

5.2.1 'Test1 Elettrico'

This model of energy district includes the following devices:

- a fuel burning generator;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- a L2 load;
- a L3 load.

The dimension of the search space is 294, i.e. there are 294 variables to be optimized, subject to 10 physical constraints and 588 bound constraints. This is the simplest example of energy district because there are no thermal configurations nor accumulators but only electric devices, few loads, so the dimension of the problem is lower then in other cases.

Table 5.2.1 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

We now present results obtained on more complex districts, with more devices and thermal configurations.

5.2.2 'Test2 Termico HOT1'

This model of energy district includes the following devices:

- an accumulator;
- a configuration HOT1;
- a wind generator;
- a photovoltaic generator;

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	27.2	0.2	27.6	26.9	1004	84	1205	881	0.5	9
SLP_1	27.8	2.6	34.4	26.6	60	4	64	52	1.4	1.4
SLP_2	26.9	0.3	27.5	26.6	51	12	73	34	1.4	1.2

Table 5.2.2: *Test2 Termico HOT1, comparison of solvers.*

- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.2 a scheme of the configuration HOT1 is depicted.

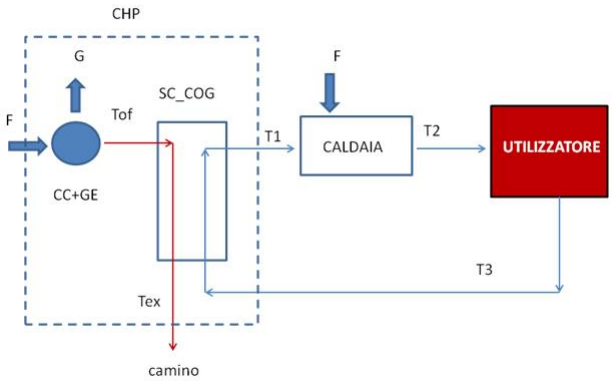


Figure 5.2.2: *Configuration HOT1: CHP and boiler.*

The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.2 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms. In Figure 5.2.3 on the left is depicted the decrease of the objective function provided by PSO algorithm, at the centre that provided by SLP_1 and on the right that provided by SLP_2. Notice that the scale on x axis in not the same in the three plots. Comparing the first figure with the other two it is evident that the convergence rate of SLP algorithm is really higher than that of PSO, that is an obvious consequence of the fact that SLP uses first order informations on f. Comparing the last two figures we can notice that SLP_1 performs more iterations than SLP_2, even if in this case the number of iterations performed by the two algorithms is not prominently different.

5.2.3 'Test2 Termico HOT2'

This example includes the following devices:

- an accumulator;
- a configuration HOT2;
- a wind generator;
- a photovoltaic generator;
- a L1 load;

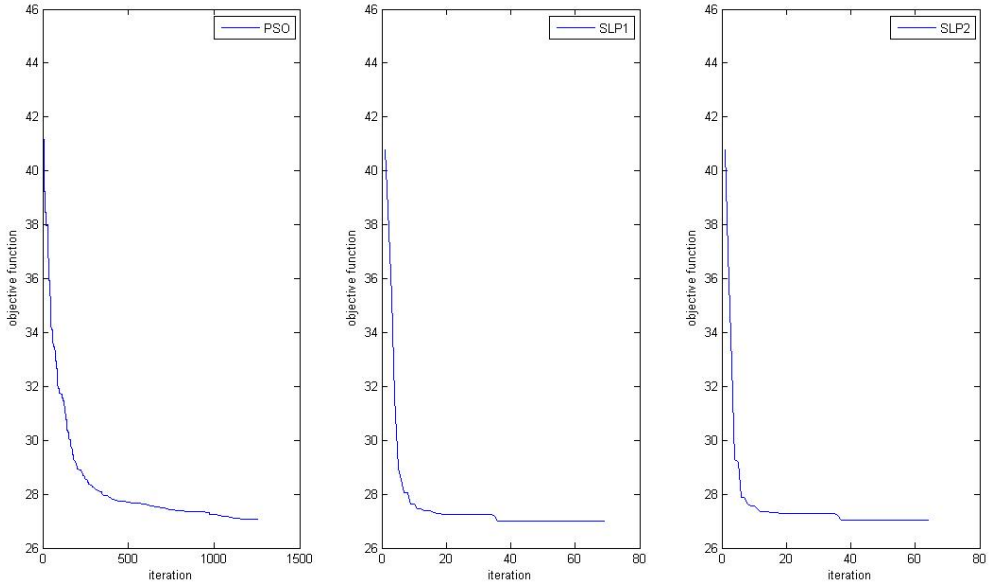


Figure 5.2.3: *Test2 Termico HOT1, decrease of the objective function provided by the three solvers.*

- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.4 a scheme of the configuration HOT2 is depicted.

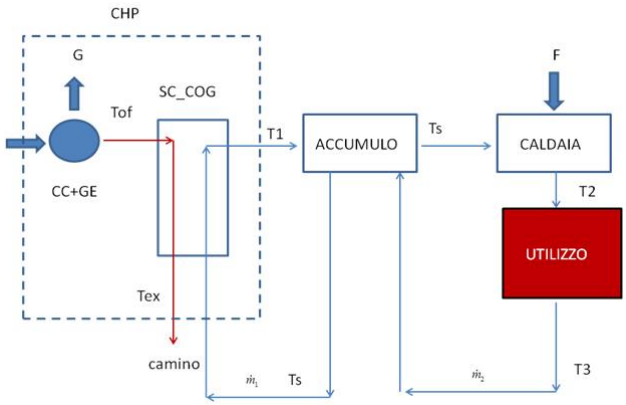


Figure 5.2.4: *Configuration HOT2: CHP assisted by boiler, with accumulator.*

The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.3 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.4 'Test2 Termico HOT3'

This example includes the following devices:

- an accumulator;
- a configuration HOT3;
- a wind generator;

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	28.0	0.2	28.3	27.7	1045	97	1165	836	0.6	10.4
SLP_1	28.3	1.3	31.4	27.4	68	7	85	63	1.4	1.6
SLP_2	27.8	0.3	28.6	27.7	52	8	65	43	1.4	1.2

Table 5.2.3: *Test2 Termico HOT2, comparison of solvers.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	27.4	0.4	28.0	26.9	1063	220	1457	587	0.3	5.3
SLP_1	28.0	2.6	34.9	26.7	64	3	70	58	0.4	0.4
SLP_2	27.2	0.3	27.8	26.8	48	7	60	38	0.4	0.3

Table 5.2.4: *Test2 Termico HOT3, comparison of solvers.*

- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.5 a scheme of the configuration HOT3 is depicted.

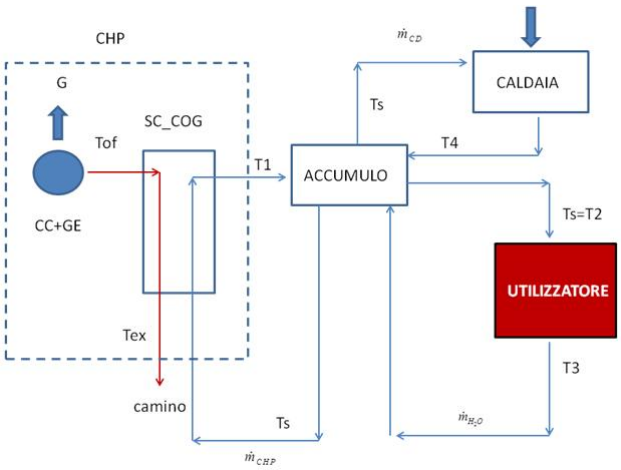


Figure 5.2.5: *Configuration HOT3: CHP connected in parallel with boiler and accumulator.*

The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.4 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.5 'Test2 Termico HOT4'

This example includes the following devices:

- an accumulator;
- a configuration HOT4;
- a wind generator;

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	30.8	1.2	32.3	28.7	1137	277	1642	660	0.4	7.6
SLP_1	27.7	0.8	28.9	26.8	70	7	87	64	0.6	0.7
SLP_2	27.9	1.2	29.9	26.8	56	11	73	34	0.6	0.6

Table 5.2.5: *Test2 Termico HOT4, comparison of solvers.*

- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.6 a scheme of the configuration HOT4 is depicted.

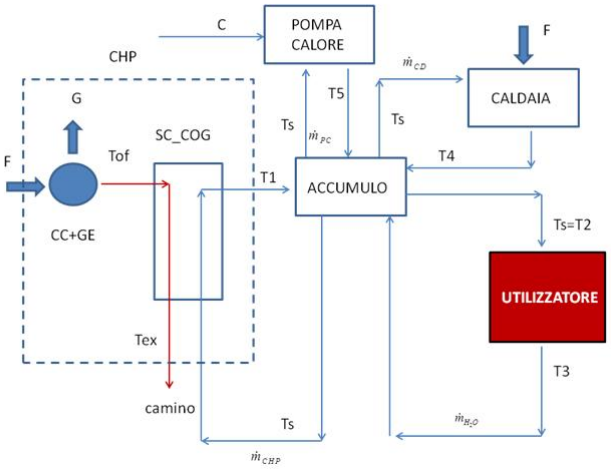


Figure 5.2.6: *Configuration HOT4: CHP, boiler and heat pump connected in parallel and accumulator.*

The dimension of the search space is 494 subject to 213 process constraints and 988 bound constraints. Table 5.2.5 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.6 'Test2 Termico HOT5'

This example includes the following devices:

- an accumulator;
- a configuration HOT5;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;

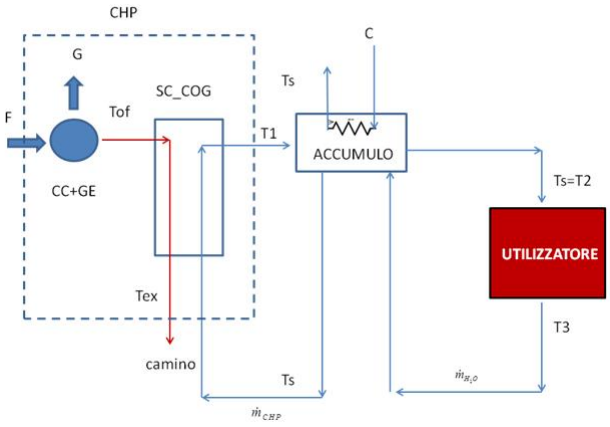


Figure 5.2.7: *Configuration HOT5: CHP assisted by electrical resistance and accumulator.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	27.6	0.4	28.3	27.1	1055	230	1414	635	0.3	5.3
SLP_1	29.7	6.8	47.7	26.8	69	5	80	63	0.4	0.5
SLP_2	27.2	0.9	29.8	26.7	53	7	62	39	0.4	0.4

Table 5.2.6: *Test2 Termico HOT5, comparison of solvers.*

- 2 L3 loads.

In Figure 5.2.7 a scheme of the configuration HOT5 is depicted.

The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.6 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.7 'Test2 Termico HOT6'

This example includes the following devices:

- an accumulator;
- a configuration HOT6;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.8 a scheme of the configuration HOT6 is depicted.

The dimension of the search space is 398 subject to 212 process constraints and 796 bound constraints. Table 5.2.7 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

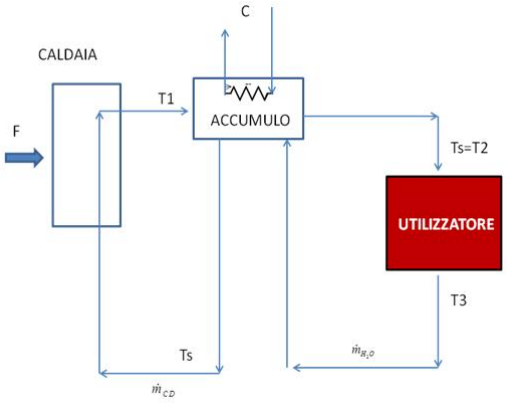


Figure 5.2.8: *Configuration HOT6: boiler assisted by electrical resistance and accumulator.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	49.7	0.9	51.8	48.6	998	256	1368	564	0.3	5.0
SLP_1	48.5	1.3	51.8	47.6	72	7	86	63	0.3	0.4
SLP_2	48.4	0.3	48.7	47.9	52	7	60	40	0.3	0.3

Table 5.2.7: *Test2 Termico HOT6, comparison of solvers.*

5.2.8 'Test2 Termico HOT7'

This example includes the following devices:

- an accumulator;
- a configuration HOT7;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.9 a scheme of the configuration HOT7 is depicted.

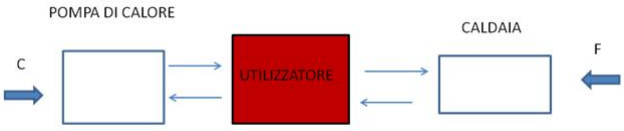


Figure 5.2.9: *Configuration HOT7: heat pump and boiler.*

The dimension of the search space is 398 subject to 212 process constraints and 796 bound constraints. Table 5.2.8 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	45.7	0.4	46.5	44.9	994	136	1172	736	0.3	4.9
SLP_1	44.4	0.5	45.9	44.2	64	7	81	58	0.3	0.3
SLP_2	44.7	1.2	47.3	43.8	45	7	54	32	0.3	0.2

Table 5.2.8: *Test2 Termico HOT7, comparison of solvers.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	45.7	0.4	46.3	45.0	1058	149	1345	855	0.3	5.3
SLP_1	44.9	0.8	46.4	44.0	61	3	67	55	0.3	0.3
SLP_2	44.4	0.4	44.9	43.7	46	16	69	32	0.3	0.2

Table 5.2.9: *Test2 Termico HOT8, comparison of solvers.*

5.2.9 'Test2 Termico HOT8'

This example includes the following devices:

- an accumulator;
- a configuration HOT8;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.10 a scheme of the configuration HOT8 is depicted.

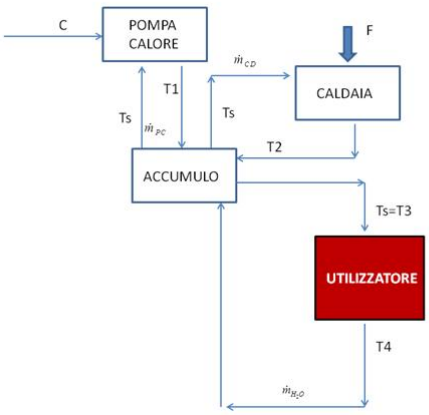


Figure 5.2.10: *Configuration HOT8: heat pump and boiler connected in parallel with accumulator.*

The dimension of the search space is 398 subject to 212 process constraints and 796 bound constraints. Table 5.2.9 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	25.2	0.2	25.4	25.0	1495	82	1634	1396	0.5	12.4
SLP_1	26.0	0.9	28.2	25.5	68	7	78	59	0.9	1.0
SLP_2	25.6	0.2	26.0	25.3	53	8	69	44	0.9	0.8

Table 5.2.10: *Test2 Termico COLD1, comparison of solvers.*

5.2.10 'Test2 Termico COLD1'

This example includes the following devices:

- an accumulator;
- a configuration COLD1;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.11 a scheme of the configuration COLD1 is depicted. The dimension of the

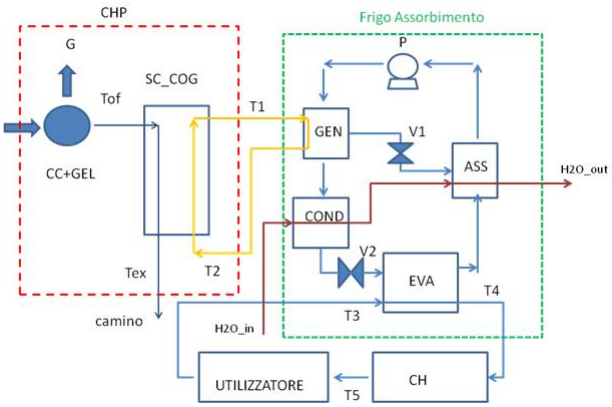


Figure 5.2.11: *Configuration COLD1: absorption refrigerator assisted by CHP and electric chiller.*

search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.10 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.11 'Test2 Termico COLD2'

This example includes the following devices:

- an accumulator;
- a configuration COLD2;
- a wind generator;

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	25.3	0.1	25.5	25.1	1513	214	1714	1238	0.5	12.6
SLP_1	26.2	1.6	30.4	25.3	68	7	82	57	0.9	1.0
SLP_2	25.6	0.4	26.4	25.2	35	2	39	33	0.9	0.5

Table 5.2.11: *Test2 Termico COLD2, comparison of solvers.*

- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure 5.2.12 a scheme of the configuration COLD2 is depicted.

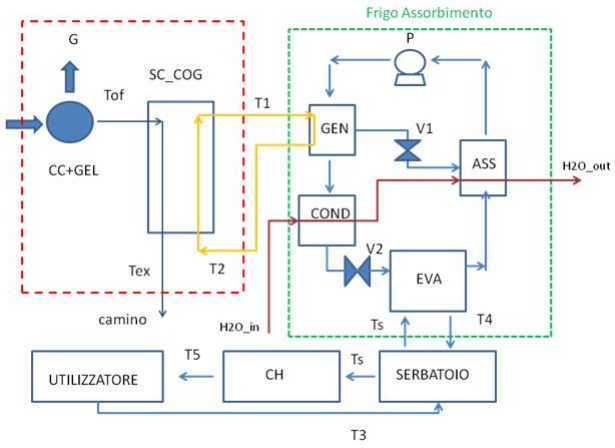


Figure 5.2.12: *Configuration COLD2: absorption refrigerator assisted by CHP, electric chiller and accumulator.*

The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.11 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.12 'Test2 Termico COLD3'

This example includes the following devices:

- an accumulator;
- a configuration COLD3;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

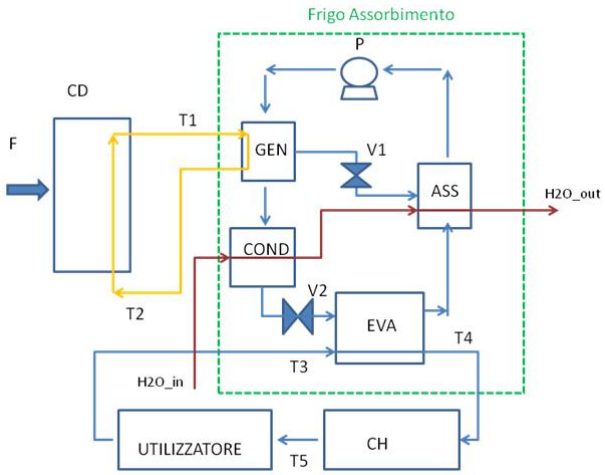


Figure 5.2.13: *Configuration COLD3: adsorption refrigerator assisted by boiler and electric chiller.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	30.6	0.1	30.7	30.4	1316	341	1797	956	0.4	8.8
SLP_1	30.0	0.3	30.3	29.5	75	10	93	60	0.5	0.6
SLP_2	30.0	0.3	30.6	29.4	47	9	60	35	0.5	0.4

Table 5.2.12: *Test2 Termico COLD3, comparison of solvers.*

In Figure (5.2.13) a scheme of the configuration COLD3 is depicted. The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.12 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

5.2.13 'Test2 Termico COLD4'

This example includes the following devices:

- an accumulator;
- a configuration COLD4;
- a wind generator;
- a photovoltaic generator;
- a L1 load;
- 2 L2 loads;
- 2 L3 loads.

In Figure (5.2.14) a scheme of the configuration COLD4 is depicted. The dimension of the search space is 398 subject to 213 process constraints and 796 bound constraints. Table 5.2.13 shows the results of the optimization process for this specific model of energy district, provided by the three algorithms.

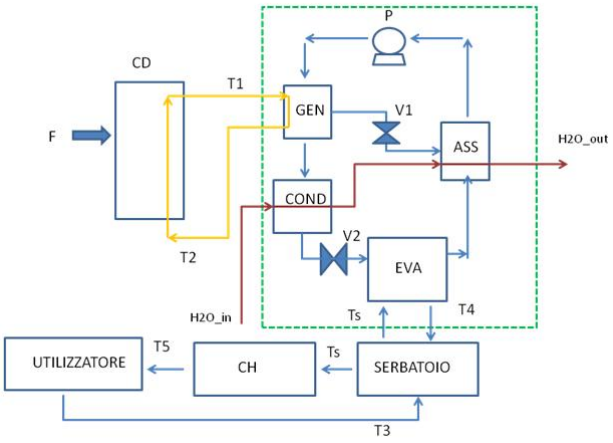


Figure 5.2.14: *Configuration COLD4: adsorption refrigerator assisted by boiler, electric chiller and accumulator.*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	31.5	0.3	31.9	31.2	1224	339	1597	642	0.4	8.2
SLP_1	30.8	0.1	31	30.6	76	11	94	61	0.6	0.8
SLP_2	30.8	0.3	31.5	30.4	50	9	68	40	0.6	0.5

Table 5.2.13: *Test2 Termico COLD4, comparison of solvers.*

5.2.14 Comments and remarks

Before discussing the obtained results, it is worth making same general comments about the different background of the two algorithms. PSO is a research algorithm and uses only information got by function evaluations, so we expect that a high number of iterations is necessary for the optimization process. SLP, on the other hand, uses also first order informations on f, so we can expect that computational costs required by the calculation of gradients are repaid with a higher rate of convergence. It is also important to notice that the software tool was designed in order to be used in conjunction with a SLP solver, so many implementation choices were made in order to fit well with this specific solver, e.g. the decisions of smoothing the θ function, of treating the discrete variables as continue ones and rounding them at the end of the optimization process, of implementing a two phase strategy to overcome problems due to a too severe restriction of the trust region due to the presence of batteries in the district. PSO solver has to be melt in the existing code, so the above approximations are made also when using this solver, even if they are unnecessary and PSO might obtain better results on the original problem. PSO is indeed capable of dealing with non-smooth functions and there exist variations of the algorithm to deal with mixed integer problems, [26]. Also the use of a two phase strategy is unnecessary when using PSO algorithm, but has been maintained to preserve the original structure of the code.

The numerical tests have confirmed our expectations about the number of iterations. The previous tables indeed shows that PSO algorithm needs more than 1000 iterations to converge, while SLP converges in less then 100 iterations, even if an iteration of SLP algorithm lasts longer, as it is shown in the column time/iter(·), because it is not always possible to calculate the gradient in an analytic way so numerical approximations using finite differences are necessary, and this is a quite heavy calculation. Nevertheless the execution times for PSO algorithm are really higher than those for SLP algorithm, so that for real time optimization an SLP approach is certainly more suitable.

From results shown above we can notice that PSO performance are not influenced by the initial swarm, in different runs results are not really different from each other and the standard

deviation on the objective function is really low in most test cases. The choice of the initial swarm may influence particles trajectories and so the way they reach the solution, but not the final result.

SLP_1 in the most part of the test cases shows an higher standard deviation on the value of objective function with respect to the other two solvers. It has indeed been observed then varying the initial guess sometimes the execution stops too early and it provides a solution with a high value of the objective function, probably because it more likely converges to a non optimal local minimum when it cannot find a feasible solution.

SLP_2, compared to SLP_1 has better performance in most test cases, the mean value of the objective function is lower and also the standard deviation. From the numerical tests it has been observed that even varying the initial guess SLP_2 is less often stuck in non optimal local minima and reaches solution approximations close to each other. Thanks to the stopping criterion we used, we can be sure that at the end of the optimization process an approximation of a stationary point for the original problem is reached.

From the numerical tests it has emerged that the mean number of iteration for SLP_2 is lower than the one of SLP_1, this is due to the fact that the stopping criterion for SLP_1 is not so satisfactory. Indeed, apart from the fact that at the end of the optimization process we have not an optimality measure of the provided solution approximation, in many test cases the execution could be stopped earlier, in fact in the last iterations no progress are made, as we can see from Figure 5.2.15. In this case it is particularly evident the different performance provided by the stopping criteria associated to SLP_1 and to SLP_2, in terms of iterations required to converge and of execution times.

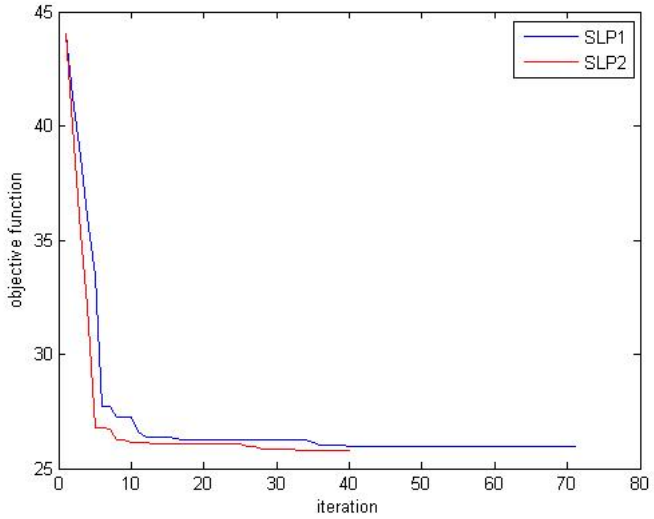


Figure 5.2.15: *SLP_1 and SLP_2: comparison of the number of iterations.*

Comparing SLP_2 and PSO we can observe that in an half of the test cases the two algorithms show almost the same performance, but in the other half of the test cases SLP_2 outperforms PSO by 3% on average with a peak of 9% in the test case 'Test2 Termico HOT4'. From these numerical results we can observe that the capacity of the evolutionary algorithm to search for the global minimum does not emerge in many test cases. We can conjecture that this fact may be due to the constraints handling strategy, because from the results of Section 5.1 we have seen that in unconstrained problems this capability emerges clearly. So as a future improvement, it would be interesting to study if an improvement of the constraints handling strategy would enhance PSO performance. Another critical aspect is the solution of stagnation problems. In fact, all the strategies we have tested, especially in highly constrained problems, were not so efficient, so also this remains an open issue for future developments.

Chapter 6

Application to a real energy district

The examples shown in Section 5.2 model several synthetic configurations of energy districts, but no one models a really existing district. In this chapter we show the results of the application of ECOS ES to a real industrial energy district located in Pisa. For the numerical experimentation we chose to simulate the winter configuration of the district. The numerical experimentation has been carried out on a PC equipped with a AMD Phenom(tm)II X4 965 Processor 3.40 GHz, 8.00 GB RAM, Windows 7 Professional 64 bit and using Matlab R2012a. Results shown in the following tables are the average of those obtained over 5 runs. For the meaning of the headings of the tables we refer to the introduction of Chapter 5. The PSO version we used is the hybrid version described in Section 4.10, and also the choices of tolerances and constants are the same made in Section 5.2: the swarm size is 20, $c_1 = 1.3$, $c_2 = 2.8$, $c_3 = 1$, for w a linearly decreasing scheme is adopted, $w_k = w_{max} - (w_{max} - w_{min})\frac{k}{k_{max}}$, with $w_{max} = 0.6$, $w_{min} = 0.1$ and $k = 1, \ldots, k_{max}$, the topology scheme adopted is a star topology, the stopping criterion is the one described in Equation (5.0.1) with a fixed tolerance $\epsilon = 10^{-3}$, the constraints handling strategy for bound constraints is the one described in Section 4.7, for nonlinear constraints is the one described in Section 4.10, with $\tau_0 = 0.1$ and $\tau_{k+1} = (1 - 0.01)\tau_k$, the regrouping mechanism described in Section 4.6 is adopted. The stopping criterion for the three algorithm are the same used in Chapter 5. For SLP algorithm we set $k_{max} = 80$ and for PSO algorithm $k_{max} = 700$, where we remind that k_{max} is the maximum number of iterations.

In this energy district no accumulators are present, so in this case the two phase strategy described in Section 1.4 is not used.

6.1 Optimization of Pisa district

To deal with this problem we had to build the model of this specific energy district, creating an Excel workbook with all the district devices. Our district model comprises:

- a configuration HOT2 with a CHP characterized by rated power 25 kWe and rated thermal power 75 kWt, a gas boiler with rated thermal power 35 kWt, a tank for the storage of hot water with capacity 9400 kj/ °C;
- a photovoltaic generator with rated power 14 kWe;
- a wind farm with rated power 3 kWe;
- a L1 load related to lighting consumptions;
- a L1 load related to heating consumptions.

Solver	$\bar{f}_0$	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time/iter(s)	time(m)
PSO	89.3	73.5	0.08	73.6	73.4	377	34	410	316	0.4	2.7
SLP_1	89.3	74.8	2.8	80.5	73.4	74	5	80	69	3.2	4
SLP_2	89.3	74.9	2.8	80.6	73.4	23	9	36	9	2.6	1

Table 6.1.1: *Results of the optimization of the energy district of Pisa.*

Among all the available hot configurations we chose the HOT2, that is the one that better fits with the heat management of the district. Then we had to enter the scalar parameters and the input tables, that have been calibrated on the measured values. The arising optimization problem has 288 variables, 576 bound constraints and 1 physical constraint on the maximum number of ignitions for CHP, as described in Equation (1.1.3). For this district we have at our disposal data referring to an unoptimized management of local resources, so that we can evaluate savings arising from the optimized management provided by ECOS package. In fact in Table 6.1.1, that shows the results of the optimization of Pisa district provided by the three algorithms, we report also the value of the unoptimized objective function f_0 that is computed using those data. We can see that in this case PSO algorithm performs better than SLP algorithm, providing a lower value of the objective function and also a really smaller standard deviation. This fact appears to confirm our conjecture presented in Section 5.2.14. In this case there is just one mild nonlinear constraint so the constraint handling strategy does not have a strong impact on the search process and the capability of the evolutionary algorithm to search for the global minimum appears as in an unconstrained test case. On the contrary the test cases presented in Sections 5.2.1 - 5.2.13 are subject to an higher number of nonlinear constraints, and comprise also loads whose constraints are particularly difficult to satisfy.

We can also notice that in this case also for PSO algorithm the execution time is quite reasonable, it is even lesser than the execution time of SLP_1, because the number of iteration PSO performs is not large and one iteration of PSO algorithm is less expensive than one iteration of SLP algorithm. Then, for this test case PSO solver could be used also in real time optimization.

Comparing the unoptimized management of local resources to the optimized one, we can see that ECOS package provides considerable saving, about 18% daily saving.

6.2 Robustness analysis

On this test case we have performed a robustness analysis of the three algorithms, varying some input parameters, namely the sell price of energy (the purchase price of energy is fixed for this specific energy district), and the thermal load Q_c for the HOT2 configuration.

As far as the price of energy is concerned, we have obtained the data from the tables of the PUN (Prezzo Unico Nazionale), choosing values referred to different past days. We have chosen 11/12/2012, 25/12/2012 that is Christmas day so prices are lesser than other days, and 8/02/2012 when prices were particularly high. The results of the previous section were obtained with prices on 11/12/2013. In Figures 6.2.1, 6.2.2 and 6.2.3 the prices trends on these days are compared with the one on 11/12/2013. Note that for PUN prices the time unit is one hour, while in ECOS ES the time unit is 15 minutes. In the previous test cases it was decided to assume for 4 consecutive time units the same value of the price, for this reason in Figures 6.2.1, 6.2.2, 6.2.3 the prices on 11/12/2013 are represented by a stairs plot, while for the other days we decided to make an interpolation of the data.

Table 6.2.1 shows the results of the optimization process of the test case of Pisa for the day 11/12/2012, Table 6.2.2 for the day 25/12/2012 and Table 6.2.3 for the day 8/02/2012, for the

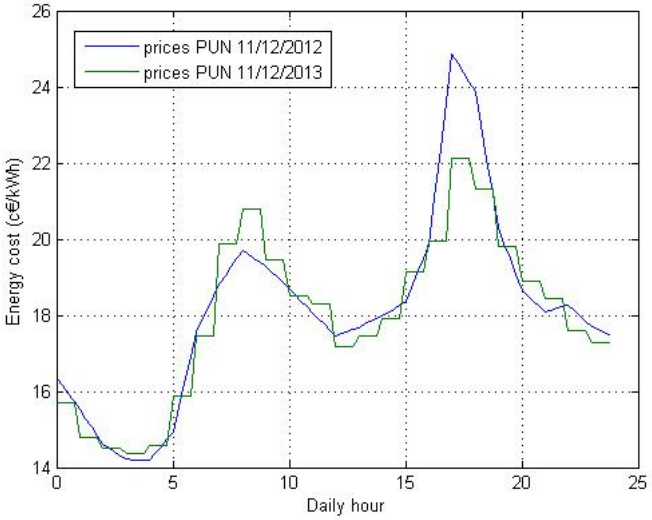


Figure 6.2.1: *Prices on 11/12/2012 and 11/12/2013*

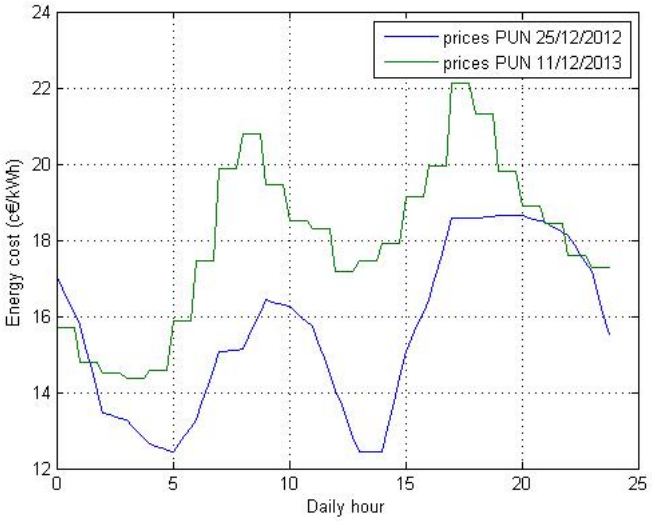


Figure 6.2.2: *Prices on 11/12/2012 and 25/12/2013*

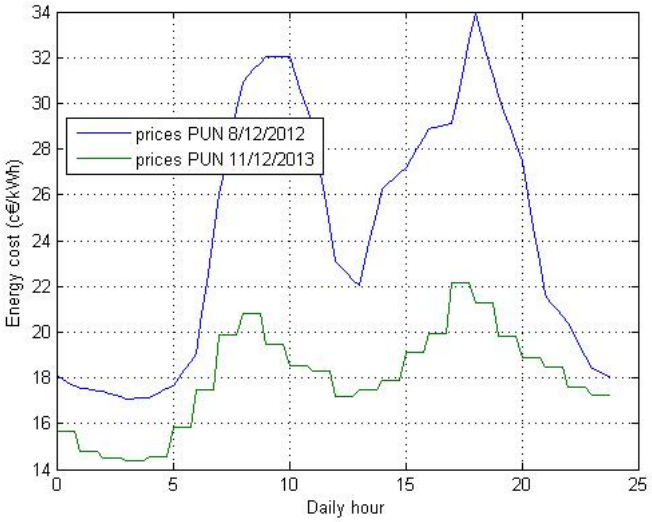


Figure 6.2.3: *Prices on 11/12/2012 and 8/02/2013*

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(m)
PSO	73.6	0.09	73.7	73.5	377	16	395	354	2.8
SLP_1	77.0	2.4	80.3	73.6	76	5	80	69	5
SLP_2	77.0	2.4	80.3	73.6	34	29	80	9	2

Table 6.2.1: *Energy district of Pisa, PUN prices on 11/12/2012,* $f_0 = 89.1$.

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(m)
PSO	72.3	0.04	72.4	72.3	399	24	435	380	2.9
SLP_1	75.7	3.6	82.6	72.3	74	7	80	65	3.4
SLP_2	75.7	3.6	82.6	72.3	20	16	45	4	1

Table 6.2.2: *Energy district of Pisa, PUN prices on 25/12/2012,* $f_0 = 82.4$.

three solvers.

As far as the thermal load is concerned, starting from the trend used for the test case described in Section 6.1, to which we will refer with Q_c and that is characterized by mean value 26, standard deviation 12, maximum 47.6 and minimum 0.3, we created new trends, enhancing the deviation from the mean value in order to have more distributed values, with more marked peaks. An example of such trends is depicted in Figure 6.2.4 on the right and it is compared to the original thermal load Q_c on the left. Tables 6.2.4 and 6.2.5 show the results of the optimization of Pisa district varying the parameter Q_c. The thermal load used to obtain results in Table 6.2.4 is characterized by: mean value 26, standard deviation 14, maximum 58.3 and minimum 0.4, while the thermal load used to obtain results in Table 6.2.5 is characterized by: mean value 29, standard deviation 16, maximum 64 and minimum 0.3.

6.3 Comments and Remarks

From the results shown in this Section we can observe that all the three algorithms are robust.

SLP_1 and SLP_2 show the same behaviour, due to the fact that the problem is subject to just one mild nonlinear constraints so the penalty function approach of SLP_2 has not a strong impact on the optimization process. However SLP_1 is more expensive because an higher number of iterations is required to converge, due to the different stopping criterion adopted.

We can notice that in many runs SLP_1, and in few runs also SLP_2, stops because the maximum number of iterations has been reached, and the stopping criterion (5.0.2) is not satisfied. We decided not to allow a larger number of iterations because from numerical test it was shown that the stopping criterion is satisfied after a quite large number of iterations, about 200, but the progress in terms of decrease of the objective function was really low, so the extra computational costs did not improve the quality of the solution approximation.

When optimizing this specific district PSO algorithm shows the best performance, providing

Solver	$\bar{f}$	σ_f	max f	min f	$\bar{k}$	σ_k	max k	min k	time(m)
PSO	77.7	0.01	77.7	77.6	376	24	407	344	2.8
SLP_1	79.2	2.9	85.0	77.6	73	6	80	67	3.5
SLP_2	79.2	2.9	85.0	77.6	52	29	80	4	1

Table 6.2.3: *Energy district of Pisa, PUN prices on 8/02/2012,* $f_0 = 106.0$.

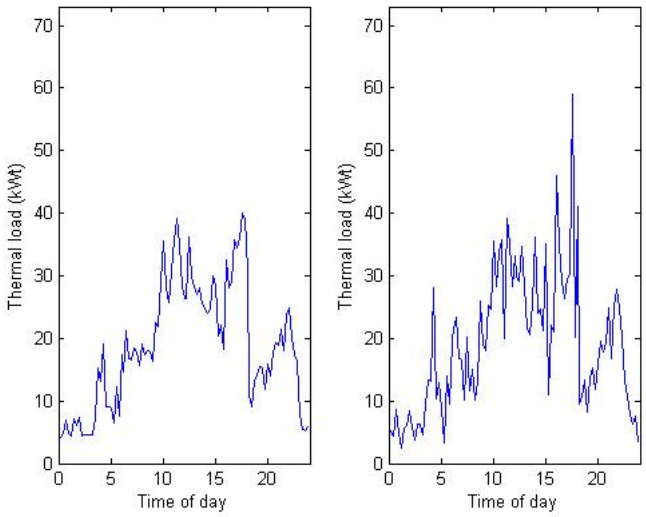


Figure 6.2.4: *An example of thermal load (right) obtained enhancing the deviation from the mean value of the original distribution (left).*

Solver	$\bar{f}$	σ_f	$\max f$	$\min f$	$\bar{k}$	σ_k	$\max k$	$\min k$	**time(m)**
PSO	73.6	0.1	73.8	73.5	383	26	416	353	2.8
SLP_1	76.3	2.6	80.6	73.5	77	7	80	63	5
SLP_2	76.3	2.6	80.6	73.5	22	15	40	14	1

Table 6.2.4: *Energy district of Pisa, variation of parameter Q_c, $f_0 = 89.6$.*

Solver	$\bar{f}$	σ_f	$\max f$	$\min f$	$\bar{k}$	σ_k	$\max k$	$\min k$	**time(m)**
PSO	73.5	0.05	73.6	73.4	346	35	400	312	2.5
SLP_1	76.6	1.7	78.7	74.7	74	7	80	66	4.4
SLP_2	76.6	1.7	78.7	74.7	44	29	80	11	1.4

Table 6.2.5: *Energy district of Pisa, variation of parameter Q_c, $f_0 = 91.0$.*

lower values of the objective function and also a lower standard deviation, outperforming SLP by $3 - 4\%$. These tests cases appears to have an higher number of local minima compared to the ones of Section 5.2, probably due to the fact that the problem is subject to less constraints. In these cases indeed SLP approaches on more runs find different solution approximations and the results show an high value of the standard deviation. In fact on each run varying the initial guess the algorithms, being designed to converge on a local minimum, converges to different points. PSO on the other hand converges to the same solution approximation, showing clearly that it has been designed to converge to a global minimum.

Bibliography

[1] N.F.Attia and Z.Michalewicz, *Evolutionary Optimization of Constrained Problems* Proceedings of the third Annual Conference On Evolutionary Programming, pp.98-108, World Scientific, 1994.

[2] N.A.Aziz, A.W.Mohemmed, M.Y.Alias, K.Ab.Aziz, *Particle Swarm Optimization for Constrained and Multiobjective Problems: A Brief Review*, 2011 International Conference on Management and Artificial Intelligence IPEDR vol.6 IACSIT Press, Bali, Indonesia, 2011.

[3] J.Born, H.Mühlenbein and D.Schomisch, *The Parallel Genetic Algorithm as Function Optimizer*, Parallel Computing, 17, pages 619–632, 1991.

[4] L.Brugnano and D.Trigiante *Solving Differential Problems by Multistep Initial and Boundary Value Methods.* Gordon and Breach Science Publ., Amsterdam, 1998.

[5] R.H.Byrd, N.I.M.Gould, J.Nocedal, R.A.Waltz, *An algorithm for nonlinear optimization using linear programming and equality constrained subproblems* , Mathematical Programming, Springer, Ser. B 100: 27-48, 2004.

[6] M.Cantù, I.Fastelli, J.Riccardi, M.Schiavetti, *Modello Matematico Ottimizzatore Distretti Energetici*, Technical Report, Enel Ingegneria e Ricerca, Pisa, 2014.

[7] P.H.Chen, *Particle Swarm Optimization for Power Dispatch with pumped Hydro*, in Particle Swarm Optimization, A.Lazinika, Ed. 2009.

[8] M.Clerc, *Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens* ,Department of Computer Science, University of Essex, Technical Report CSM-460, August 2006.

[9] M.Clerc, *Particle Swarm Optimization*, ISTE Ltd, 2006.

[10] G.Coath and S.K.Halgamuge, *A Comparison of Constraint-Handling Methods for the Application of Particle Swarm Optimization to Constrained Nonlinear Optimization Problems*, Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, p. 2419 - 2425 Vol.4 , 2003.

[11] A.R.Conn, N.I.M.Gould, P.L.Toint, *Trust-Region methods*, Society for Industrial e Applied, 2000.

[12] R.C.Eberhart and J.Kennedy, *Particle Swarm Optimization*, Proceedings of the IEEE International Conference on Neural Networks, IEEE Press, pp. 1942–1948, 1995.

[13] R.C.Eberhart and J.Kennedy, *A new optimizer using particle swarm theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), pp. 39-43, 1995.

[14] R.Eberhart and Y.Shi, *Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization* Proc. Congress on Evolutionary Computation, La Jolla, CA, pp. 84-88, 2000.

[15] R.Eberhart and Y.Shi, *A modified particle swarm optimizer*, Proceedings of the 1998 IEEE International Conference of Evolutionary Computation, 1998.

[16] G.I.Evers *An automatic regrouping mechanism to deal with stagnation in particle swarm optimization*, M.S. thesis, The University of Texas – Pan American, Edinburg, TX, 2009.

[17] W.Ferrara, J.Riccardi, S.Sello, *Enel Customer Optimization Service Expert System Development*, Technical Report, Enel Ingegneria e Ricerca, Pisa, 2014.

[18] R.Fletcher, *Practical Methods of Optimization*, 2nd edition, Wiley, Chichester, 1987.

[19] R.Fletcher, E.Sainz de la Maza, *Nonlinear programming and nonsmooth optimization by successive linear programming*, Mathematical Programming 43: 235-256, 1989.

[20] D.E.Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley Company, January 1989.

[21] U.Grenander and F.Heppner, *A stochastic nonlinear model for coordinated bird flocks*, in S. Krasner, Ed., The Ubiquity of Chaos, AAAS Publications, Washington, DC, 1990.

[22] T.Hatanaka, T.Korenaga, N.Kondo and K.Uosaki, *Search Performance Improvement for PSO in High Dimensional Space*, in Particle Swarm Optimization, Edited by Aleksandar Lazinica, 2009, In-tech Ed.(sistemare)

[23] R.L.Haupt and S.E.Haupt, *Practical Genetic Algorithms* Wiley-Interscience, 2004.

[24] C.Houck and I.Joines, *On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's*, Proceedings of the First IEEE Conference on Evolutionary Computation, vol. 2, pp. 579-584, June 1994.

[25] D.Huang, Y.Jin, B.Liu, F.Tang and L.Wang, *An improved particle swarm optimization combined with chaos* Chaos, Solition and Fractals, vol. 25, pp. 1261-1271, 2005.

[26] S.Kitayama and K.Yasuda, *A method for mixed integer programming problems by Particle Swarm Optimization*, Electrical Engineering in Japan, Volume 157, Issue 2, pages 40–49, November 2006.

[27] A.Lazinica, *Particle Swarm Optimization*, 2009.

[28] G.M.Mancuso, G.Pannocchia, *Steady state optimizer of energy district,* Technical Report, Dipartimento di Ingegneria Civile e Industriale, Università di Pisa, 2014.

[29] Z.Michalewicz, M.Schoenauer *Evolutionary Algorithms for Constarined Parameter Optimization Problems*, Evolutionary Computation 01/1996; 4:1-32, 1996.

[30] J.Nocedal and S.J.Wright, *Numerical Optimization*, Springer, 1999.

[31] K.E.Parsopoulos, M.N.Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*, Information Science Reference, 2010.

[32] K.E.Parsopoulos and M.N.Vrahatis, *Particle Swarm Optimization Method for Constrained Optimization Problems*, Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies, pp. 214-220, IOS Press (Frontiers in Artificial Intelligence and Applications series, Vol. 76), 2002.

[33] L. A.Rastrigin, *Systems of extremal control*, Nauka, Moscow, 1974.

[34] C.W.Reynolds, *Flocks, herds and schools: a distributed behavioural model*, Computer Graphics, 21(4):25-34, 1987.

[35] S.Sello, *Algoritmo di ottimizzazione PSO - Particle Swarm Optimization*, Technical Report, Enel Ingegneria e Ricerca, Pisa, 2014.

[36] I.C.Trelea, *The particle swarm optimization algorithm: convergence analysis and parameter selection*, Information Processing Letters 85, 317–325, 2003.

[37] E.O.Wilson, *Sociobiology: The new synthesis*, Belknap Press, Cambridge, hlA, 1975.