POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Valutazione sperimentale di formulazioni DCOP per la gestione dell'energia in piccoli complessi o edifici commerciali

Relatore: Prof. Francesco AMIGONI

Tesi di Laurea Magistrale di:
Tomas Toffolet
Matr.814293

Anno accademico 2014/2015

SOMMARIO

Il risparmio energetico è un tema che sta catturando sempre maggior interesse a livello mondiale ed è attualmente uno degli argomenti scientifici più dibattuti, sia a causa del continuo innalzamento dei costi dell'energia che per il progressivo abbandono dei programmi nucleari ma anche per i problemi ambientali causati dalle forme non rinnovabili di energia.

Questa tesi si colloca dunque nell'ambito dell'utilizzo dei sistemi multiagente (MAS) per il problema di gestione della domanda di energia elettrica, un paradigma che ha lo scopo di ottimizzare il più possibile l'uso, la gestione e il trasporto dell'energia, allo scopo di diminuire il più possibile gli sprechi.

In particolare, lo scopo di questa tesi è di valutare in maniera sperimentale l'applicabilità della formulazione DCOP (Distributed Constraint Optimization Problem) a problemi di gestione dell'energia elettrica in ambito di piccoli complessi o edifici commerciali, con agenti che rappresentano il carico di una singola area (o di un singolo appartamento) e che comunicano fra loro al fine di ottimizzare l'utilizzo della potenza totale disponibile, minimizzando i costi e garantendo di conseguenza benefici per l'intera rete elettrica.

In seguito allo sviluppo del sistema, lo abbiamo testato per mostrare quale algoritmo di risoluzione dei problemi DCOP fosse più funzionale e come cambia la risposta del sistema variando il numero di appartamenti o le variabili presenti nel problema.

Indice

Elenco delle figure

1 - Introduzione

Negli ultimi anni il tema della sostenibilità dell'energia sta catturando grande interesse a livello mondiale. Ad oggi, circa l'85% del fabbisogno energetico mondiale è coperto dai combustibili fossili [9], ma la limitata disponibilità di tali fonti di energia, insieme ai noti danni ambientali causati dall'utilizzo di esse, spinge l'umanità a ricercare nuove fonti energetiche, in particolar modo di tipo rinnovabile.

Per cercare quanto meno di ridurre l'uso di energia non rinnovabile e ottimizzare i consumi sono state ideate le smart grid: si tratta dell'insieme di due reti, una di informazione e una di distribuzione di energia elettrica. Tale nuova rete è in grado di gestire in maniera intelligente l'uso dell'energia, ottimizzandone nel frattempo il trasporto verso gli utenti finali.

Un aspetto importante delle smart grid è quello di evitare dei picchi di domanda di corrente elettrica, che causano un aumento di emissioni di CO_2 e un consumo maggiore di energia.

Per fare questo, le smart grid sfruttano il Demand Side Management (DSM), che consiste in metodi per modificare la domanda di energia da parte del consumatore, generalmente spostandone, ove possibile, la richiesta al di fuori dei periodi di punta.

Queste problematiche possono essere affrontate con un approccio centralizzato oppure con un approccio distribuito. Sebbene gli approcci centralizzati siano più semplici da implementare, l'utilizzo di approcci distribuiti garantisce numerosi potenziali vantaggi, tra cui maggiore flessibilità, affidabilità e tolleranza ai guasti.

Tra gli approcci distribuiti, quello che sta ottenendo più successo e sempre maggior interesse è il paradigma dei sistemi multiagente (MAS).

Lo scopo di questa tesi è di valutare in maniera sperimentale l'applicabilità della formulazione DCOP (Distributed Constraint Optimization Problem) a problemi di gestione dell'energia elettrica in ambito di piccoli complessi o edifici commerciali, con agenti che rappresentano il carico di una singola area (o di un singolo appartamento) e che comunicano fra loro al fine di ottimizzare l'utilizzo della potenza totale disponibile, minimizzando i costi e garantendo di conseguenza benefici per l'intera rete elettrica.

Fino ad oggi i lavori principali presentati in letteratura (come ad esempio [1], [3] ed [8]) riguardano l'ottimizzazione dei consumi a livello di singole abitazioni (livello che

chiameremo bottom), con algoritmi che cercano di minimizzare il costo necessario per raggiungere determinati obiettivi (imposti ad esempio dall'utente) rispettando determinati vincoli (ad esempio la potenza massima disponibile istantaneamente) oppure che cercano di trovare il giusto compromesso tra obiettivi diversi.

L'obiettivo che ci poniamo con questo lavoro è invece un'ottimizzazione di livello più esteso (che chiameremo top), in cui si ottimizzano i consumi di più abitazioni raggruppate in un unico complesso come un condominio oppure un residence.

In Figura 1.1 è rappresentato un esempio di livello bottom, mentre in Figura 1.2, è rappresentato lo schema concettuale del livello top.

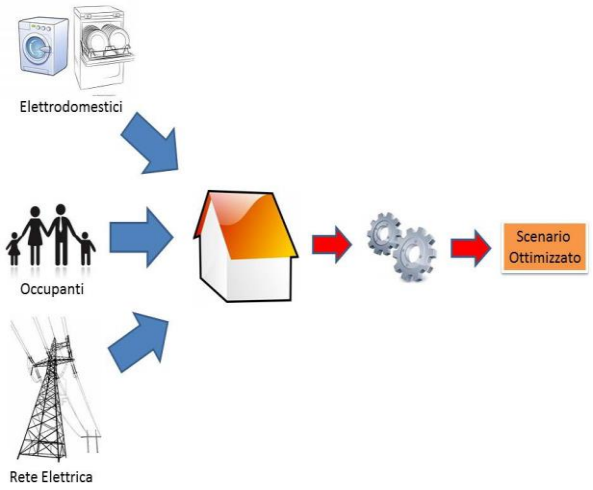


Figura 1.1 Scenario iniziale, livello bottom

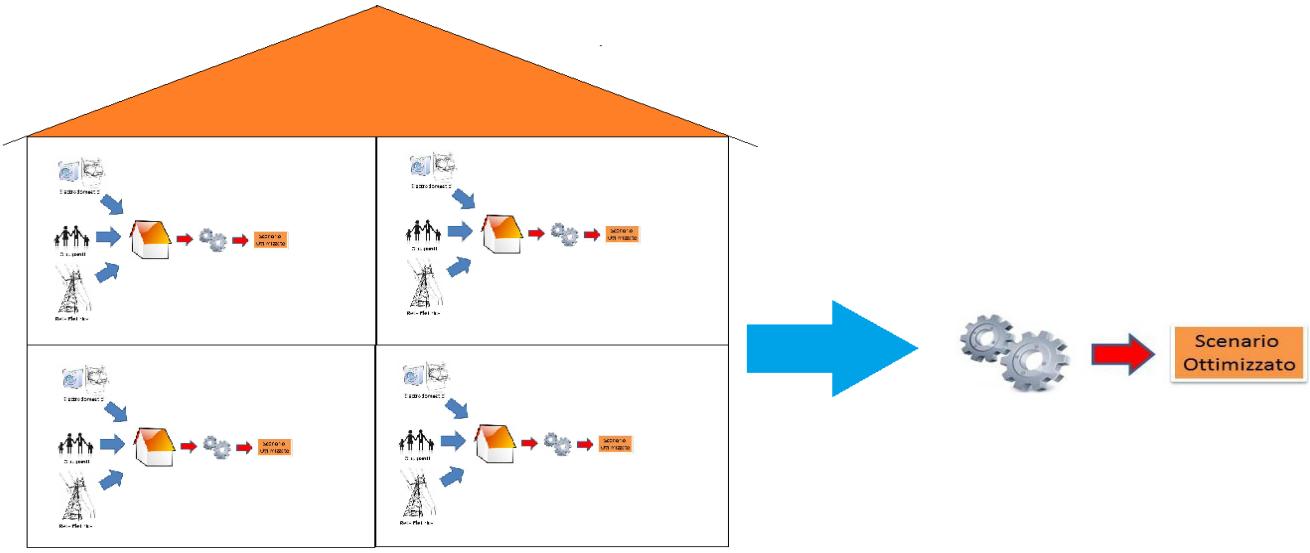


Figura 1.2 Scenario evoluto, più livelli bottom ed un livello top

I problemi a livello bottom possono essere convenientemente modellati come DCOP, quello a livello top come un DCOP oppure un più semplice DCSP (Distributed Constraint Satisfaction Problems), preoccupandoci cioè solamente di non violare determinati vincoli, come quello di potenza massima.
Le motivazioni per una tale ottimizzazione di alto livello possono essere molteplici, ad esempio la necessità di rispettare i vincoli (si ha una potenza massima disponibile per l'intero edificio), un'ottimizzazione allo scopo di minimizzare la potenza dissipata (ad esempio perché il condominio possiede dei pannelli solari e l'energia che risparmia la può reimmettere nel circuito elettrico, ottenendo un guadagno o semplicemente allo scopo di spendere meno), oppure si potrebbe voler ottenere un giusto compromesso tra risparmio e benessere dei singoli utenti.
In questo lavoro il problema verrà discusso dal punto di vista della gestione dell'energia elettrica, ma gli stessi principi possono essere riutilizzati e riadattati in altri ambienti come ad esempio la pianificazioni di itinerari, l'allocazione di risorse o in qualsiasi campo in cui vi deve essere cooperazione e coordinamento di sistemi multi agente.
Un'altra cosa importante da ricordare ai fini della trattazione è che il sistema non verrà mai visto come un "gioco", quindi la interazione tra agenti non sarà di tipo competitivo e strategico (o meglio un agente non fa ragionamenti del tipo: aziono in un momento diverso rispetto al mio vero ottimo, in modo tale da guadagnarci di più costringendo l'altro agente a scelte diverse), ma sempre e solo di tipo collaborativo.
Abbiamo quindi sviluppato e ideato alcune formulazioni DCOP, discusse in seguito e abbiamo scelto la più promettente da implementare.
Uno dei vantaggi di adottare la formulazione DCOP è il disaccoppiamento tra i modelli impiegati e gli algoritmi di risoluzione.
Ci siamo dunque serviti di FRODO, un framework open-source ideato per la risoluzione di DCOP, che simula un sistema multiagente distribuito in una macchina virtuale Java, come strumento di risoluzione dei nostri modelli DCOP (sia di livello bottom che di livello top).

La tesi è strutturata nel seguente modo.
Nel Capitolo 2 vengono passati in rassegna alcuni lavori precedenti in materia di gestione dell'energia attraverso modelli DCOP.

Nel Capitolo 3 viene presentato il framework FRODO che sarà utilizzato per la risoluzione dei modelli DCOP durante l'arco di tutto il lavoro di tesi.
Nel Capitolo 4 viene presentata una prima impostazione concettuale del problema mentre al Capitolo 5 il problema viene formalizzato.
Nel Capitolo 6 vengono presentati una serie di test effettuati con l'applicazione sviluppata a partire dal modello formalizzato ed eseguiti con gli algoritmi presentati.
Nel Capitolo 7 sono riassunti gli obbiettivi di questa tesi, le valutazioni del lavoro svolto e i possibili risvolti futuri.
In Appendice A viene mostrato un esempio di output del sistema implementato.

2 – Stato dell'arte

Questo capitolo ha l'obiettivo di presentare alcuni lavori che cercano di risolvere il problema di ottimizzazione dei consumi di edifici con algoritmi di tipo distribuito su una rete composta da più appartamenti o ambienti (come ad esempio nel caso di uno stabile).

Fino ad oggi sono stati affrontati in modo più assiduo i problemi sul singolo appartamento e con algoritmi centralizzati, ma seguendo tale strategia si rischia di avere picchi di consumo sulla rete (se tutti applicassero l'ottimizzazione singolarmente, senza curarsi degli altri), inoltre si potrebbe ottenere un risparmio maggiore con un'ottimizzazione condivisa.

Un esempio di algoritmo centralizzato è presentato in [8] e nella Sezione 2.7.

Un esempio di algoritmo distribuito invece, oltre a quelli implementati in FRODO, è presentato in [2] e nella Sezione 2.2.

Quelli che seguono sono dunque esempi di lavori già svolti che risolvono in parte il problema di un'ottimizzazione distribuita o che propongono un modello o una formalizzazione che potrebbe essere utilizzata per il medesimo scopo.

2.1 - Agent-Based Control for Decentralised Demand Side Management in the Smart Grid [1]

Il tema base di questo lavoro è l'uso di contatori casalinghi intelligenti applicati ad ogni abitazione, allo scopo di ottimizzare il consumo dei diversi apparecchi elettronici e del riscaldamento elettrico.

Il grosso problema da affrontare è il coordinamento tra i diversi contatori: infatti senza un meccanismo che li regoli, le soluzioni di ottimizzazione dei diversi apparecchi elettronici possono essere simili (se non identiche), con un relativo picco di potenza richiesta in determinati momenti, causando perdita di efficienza, maggiori emissioni di CO_2 nell'atmosfera e nei casi più gravi portare addirittura a black out.

Il lavoro si pone dunque l'obiettivo di introdurre un nuovo modello DDSM (Decentralised Demand Side Management) che ha lo scopo di gestire in maniera

decentralizzata il differimento dei carichi. Questo porta a riduzioni di costo ed emissioni, garantendo nel contempo la continuità di energia.

Ad oggi è l'utente stesso a gestire le proprie utenze (nella speranza che limiti i consumi), e può accendere qualsiasi dispositivo nel momento che preferisce (senza ovviamente superare la soglia massima di potenza utilizzabile), le tecnologie DSM invece gestiscono l'avvio di elettrodomestici, invogliando gli utenti ad utilizzare i sistemi fuori dall'orario di punta. Il secondo scopo di tale lavoro è dimostrare che tali sistemi sono efficaci anche in caso di grossi complessi e numerosi appartamenti.

Per la costruzione del sistema gli autori si sono basati su altri paper, principalmente teorici, i quali o prevedono forme di soluzioni chiuse e che hanno bisogno di un modello del comportamento possibile degli attori coinvolti nel sistema, oppure sistemi che hanno device per immagazzinare energia e usarla in momenti più opportuni, ma che non affrontano il problema dei carichi differibili e del comfort casalingo.

E' ovvio comunque che esistono dei carichi effettivamente differibili (ad esempio l'accensione di lavatrice e lavastoviglie) ed altri carichi non differibili (come l'accensione del forno o di una televisione). I contatori intelligenti devono quindi agire solo su quelli differibili, che ad oggi sono circa il 20% delle utenze (ed in crescita), senza interferire con lo stile di vita delle persone.

I carichi differibili generalmente, in ambiente domestico, si dividono in 4 categorie: strumenti per il lavaggio (ad esempio lavastoviglie o lavatrice), strumenti di raffreddamento (ad esempio il frigorifero), strumenti per scaldare l'acqua (ad esempio boiler elettrici) ed infine strumenti per riscaldare gli spazi (come radiatori o pompe per il calore).

E' chiaro come questi apparecchi, pur essendo tutti carichi differibili, siano molto diversi tra loro, ad esempio la lavatrice sottostà a timer impostati o all'accensione da parte dell'utente, mentre il riscaldamento è più legato alla temperatura della stanza.

Proprio per questo nel paper viene fatta un'ulteriore divisione tra i carichi (importante precisare che il tempo viene discretizzato campionandolo ogni mezz'ora lungo l'arco di una giornata, dando luogo ad un periodo temporale $T = \{1 \ldots 48\}$):

- Carichi differibili statici, il cui set è indicato come $l \in L$. Sono apparecchi con tempi di accensione e periodi di funzionamento determinati in modo preciso. Ogni carico possiede un parametro $o_l \subseteq T$, che rappresenta il set di time slot in

cui l'utente ha impostato l'accensione dell'apparecchio. Ogni deviazione da tale set comporta un disagio per l'utente ed è rappresentata da $d_l \in \{-24, -23, ..., +23, +24\}$ che rappresenta il discostamento dal valore o_l, o in altre parole, di quante unità temporali ci si discosta dal tempo di accensione impostato dall'utente

Tali carichi, nel nostro problema, verranno quindi modellati come una minimizzazione del disagio per l'utente causato da un possibile spostamento dell'orario di avvio ($c_l = \Delta c_l * |d_l|$, dove c_l è il costo di uno spostamento di un'unità) sommato al costo di lavoro del carico (tempo di lavoro per il costo unitario dell'elettricità).

Quindi si cerca un assegnamento tale da risparmiare il più possibile creando meno disagio all'utente.

- Carichi termici: molto più legati alla temperatura della stanza. Si cercherà di raggiungere la temperatura desiderata dall'utente minimizzando il più possibile i costi.

  La temperatura interna al tempo t (τ^t_{in}) sarà pari alla temperatura al tempo t-1 sommata ad un valore che rappresenta il riscaldamento attuato dal tempo t-1 fino al tempo t.

  Si calcolerà il costo del comfort al tempo t come: $C^t_{on} * (\tau^t_{in} - \tau_{opt})^2$, dove C^t_{on} vale 1 quando l'utente imposta la sua presenza nell'abitazione a quell'istante t, 0 negli altri casi e τ_{opt} è la temperatura desiderata dall'utente.

  La formulazione per tali carichi termici sarà quindi una minimizzazione del costo totale del comfort (somma su tutti gli istanti di tempo t del costo del comfort al tempo t) sommato al costo di utilizzo (calcolato come il prodotto tra il tempo di utilizzo e il costo unitario dell'elettricità, in modo simile al caso di carichi differibili statici).

La formulazione finale sarà quindi una minimizzazione composta dalla somma di carichi differibili statici, carichi termici e carichi fissi e non spostabili (rappresentati come dei valori $fixed_{t,i}$).

Un altro punto analizzato in questo paper è il prezzo dell'energia: in base alla politica scelta, prezzo fisso, TOU (time of use, una tariffa a periodi che cambia in base a ora,

giorno e periodo dell'anno) oppure RTP (real time pricing, che usa il prezzo reale dell'energia in quel momento) cambia il risparmio energetico, perché l'algoritmo è più o meno invogliato a modificare gli orari di accensione dei diversi apparecchi. In generale è dimostrato che la tariffa migliore per ottimizzare i consumi è quella RTP.

Il risultato finale è quindi un sistema che sfrutta la differibilità dei carichi, cercando di minimizzare i costi tramite un meccanismo AM (Adaptive Mechanism), che sfrutta il WHLM (Widrow-Hoff Learning Mechanism) per modificare gradualmente l'avvio dei carichi differibili per ogni agente (dove un agente è il singolo appartamento). In secondo luogo AM modella ulteriormente ogni agente in base a quanto spesso riadatta il suo profilo di riscaldamento.

Si nota come maggiore è la variabilità dei prezzi per l'energia e maggiore sarà il guadagno (quindi come RTP sia la tariffa migliore e perciò quella sfruttata per le simulazioni).

Oltre a ciò viene mostrato come, con l'aumento di carichi spostabili e di riscaldamento elettrico (cose plausibili in futuro), grazie a questa ottimizzazione il risparmio energetico aumenti ulteriormente mentre le emissioni di carbone calino.

Coi dati odierni il guadagno elettrico usando tale sistema è del 17%, mentre il calo di emissioni di CO_2 è del 6% circa. La simulazione riguarda un modello di 5000 abitazioni, usando profili di carico medi (invernali) di 26 mila abitazioni inglesi, con una confidenza del 95% ed esperimenti ripetuti 100 volte.

Appare subito chiara una grossa differenza con quanto vorremmo noi sviluppare: infatti nel nostro lavoro non ci preoccuperemo del tipo di carico ma considereremo tutti i carichi che si possono spostare e che fanno parte della rete uguali, focalizzando principalmente l'attenzione su apparecchi quali lavatrici o lavastoviglie e tralasciando la parte di riscaldamento elettrico. Inoltre non cercheremo di minimizzare il disagio per l'utente, in quanto è la rete dell'utente stesso a restituire un numero finito di valori accettabili dall'utente (risultati delle reti bottom), minimizzando già in partenza il proprio disagio.

Infine l'idea di usare due livelli diversi di ottimizzazione non è contemplata in questo lavoro, mentre al contrario è alla base del nostro.

2.2 - An Optimal Distributed Constraint Optimisation Algorithm for Efficient Energy Management [2]

In questo lavoro vengono approfonditi i possibili benefici derivanti dalla liberalizzazione del settore elettrico in Europa, che ha creato un tipo di mercato in cui utenti e fornitori possono operare al fine di migliorare le proprie esigenze economiche (il cliente spendere meno, il fornitore guadagnare il più possibile).

Con questo nuovo tipo di mercato l'utente può stringere accordi con diversi provider (anche più di uno contemporaneamente), in modo da ottenere i migliori prezzi possibili.

Fino ad oggi l'unica preoccupazione in questo settore era quella di gestire un singolo utente, il quale deve compiere determinate scelte allo scopo di schedulare nel miglior modo possibile l'utilizzo degli apparecchi, come detto anche in precedenza.

Invece in questo paper ci si è preoccupati del caso in cui esista una rete di utenti, ognuno dei quali deve ottimizzare i consumi, con i fornitori che reagiranno di conseguenza. Serve dunque un accordo ed una comunicazione tra i vari agenti allo scopo di migliorare i consumi per ogni utente. Invece il fornitore varierà la propria offerta in modo da migliorare il proprio guadagno e di non avere picchi di carichi nella rete, tenendo in considerazione le varie schedulazioni.

A tale scopo è stato messo a punto l'algoritmo COBB (Constraint Optimisation By Broadcasting).

La base dell'algoritmo COBB sono i problemi DCOP, adattati però a particolari richieste non convenzionali: l'algoritmo deve essere infatti completo, distribuito e deve trovare una soluzione velocemente e nel modo migliore possibile.

Come detto inoltre, l'utente può scegliere tra i diversi provider l'assegnamento migliore per ogni slot temporale (creando di fatto un modello base simile al "problema dello zaino").

Lo svantaggio di tale approccio chiaramente è la complessità, inoltre è inutile trovare un assegnamento teoricamente ottimo ma che i diversi provider non riescono a garantire (perché ad esempio viene richiesta troppa potenza ad un solo provider in un determinato istante).

Quindi il primo scopo dell'algoritmo è ottenere un ottimo condiviso da tutti gli utenti mentre il secondo è quello di permettere il funzionamento di tutti gli apparecchi elettronici che richiedono energia senza causare blackout.

In COBB si evidenziano tre punti fondamentali:

- Il mercato dell'energia, che interessa sia clienti che fornitori. Infatti il cliente dovrà avere più contratti con i diversi fornitori che specificano anche gli orari di possibile utilizzo. Tali contratti non devono per forza essere esclusivi, infatti il cliente potrà avere più contratti per gli stessi orari.
- Il sistema multiagente, in cui ogni agente corrisponde ad un cliente privato. Per semplicità ogni agente avrà un task di consumo.
- Al contrario delle aste, dove chi offre è in competizione con gli altri, gli agenti qui collaborano per raggiungere un obiettivo comune.

COBB, che necessita di lavorare con soluzioni complete e con una funzione di costo globale, funziona tramite raffinamenti successivi della soluzione.

COBB parte con una soluzione candidata per ogni agente. A questo punto l'algoritmo, eseguito da ogni agente, cerca di migliorare tale soluzione con un cambio locale dei tempi di avvio dei carichi (possono esserci più possibili scelte): se ci riesce invia in broadcast queste nuove soluzioni, altrimenti invia la soluzione iniziale. I vari agenti ricevono tali risultati e rieseguono iterativamente la stessa cosa, controllando se un cambio locale potrebbe migliorare la soluzione.

Quando l'agente riceve poi una risposta, confronta la soluzione migliore attuale con ogni singola soluzione ricevuta e tiene quella con costo inferiore.

Di seguito, in Figura 2.1 uno pseudocodice semplificato dell'algoritmo:

Essendo completo, bisognerà esplorare tutte le soluzioni, perché candidati che sembrano peggiori, potrebbero rivelarsi poi in realtà soluzioni ottime.

La complessità di COBB è 2^n*m, dove n è il numero di agenti ed m il numero di possibili alternative (ad esempio in questo caso, il problema sarà schedulato sulla giornata, divisa in ore, e quindi m varrà 24, dato che i possibili time slot di avvio sono 24 per ogni carico).

Nel paper è presentato inoltre un confronto di COBB (basato sul gioco delle regine, è stato scelto tale problema perché esistono vincoli per ogni coppia di agenti) con l'algoritmo di Asynchronous Backtraking (AB).

COBB si dimostra migliore in tutti i casi, infatti riesce a finire sempre nel tempo limite di 1000 cicli, al contrario di AB che comunque, quando termina, mediamente impiega un numero di cicli maggiori.

```
COBB(candidate_solution) {
        new_sol_list = improve (candidate solution)
        if (new_sol_list != null) {
                foreach new_solution in new_sol_list {
                        answers = broadcast (new_solution)
                }
        }
        else
        {
                answers = broadcast (candidate_solution)
        }
        sort_desc (answers)
        foreach solution in answers {
                if (fitness(solution) < fitness(best_solution))
                {
                        best_solution = solution
                }
        COBB(solution)
        }
}
```

Figura 2.1: pseucodice COBB

In particolare le prove sono state fatte con un numero n di regine variabile tra 10 e 50:

- Con n pari a 10 sia AB che COBB terminano col 100% della probabilità, ma il primo impiega mediamente 105 cicli, mentre il secondo solo 15.
- Con n pari a 50 AB termina il 50% delle volte, con numero di cicli pari a 325, mentre COBB termina il 100% delle volte con mediamente 35 cicli.

Gli autori hanno tenuto in considerazione solo il numero di cicli e non il tempo in valore assoluto, questo perché è difficile quantificare in una simulazione il tempo di broadcast, implicito in COBB.

Vi è inoltre un confronto con il noto algoritmo ADOPT (opportunamente modificato per il problema delle regine), il quale si dimostra migliore nei casi in cui il numero di regine è basso (inferiore a 8). In questo esperimento vengono usati come metrica il numero di messaggi scambiati ed il numero di volte che viene chiamata la funzione di costo globale per valutare l'assegnamento trovato (per capire se è ottimo) ed i test sono stati ripetuti 100 volte per ogni n (con n da 5 a 8)

Si è ripetuto poi la prova con n>= 8 e sia il numero di messaggi che il numero di chiamate alla funzione di costo hanno risultati peggiori nel caso di ADOPT rispetto a COBB.

Lo stesso ADOPT si dimostra inutilizzabile e con un numero di messaggi che cresce in modo quadratico al crescere del numero di regine. Al contrario COBB, pur essendo

peggiore con n<8, ha una complessità che cresce in modo lineare e quindi migliora rispetto ad ADOPT col crescere di n.
Il problema più grosso che potrebbe portare in seno COBB è quello del costo dei messaggi broadcast, che potrebbe essere lieve in caso di problemi DCSP (in quanto ci si ferma alla prima soluzione trovata), ma più consistente in caso di DCOP.
Quindi tale algoritmo è un primo tentativo di unire più utenze allo scopo di ottenere una ottimizzazione migliore rispetto al caso di ottimizzazione singola per ogni appartamento e allo scopo di dare uno strumento ai fornitori tale da poter gestire meglio le utenze ed i costi.
Anche in questo caso la differenza sostanziale dal nostro lavoro è la non differenziazione in due livelli: i vari utenti vengono visti come dei semplici agenti che richiedono energia.

2.3) TESLA: an extended study of an energy-saving agent that leverages schedule flexibility [3]

TESLA è il progetto che da certi punti di vista più si avvicina al tipo di lavoro che stiamo cercando di sviluppare. In tale sistema si sfrutta la flessibilità degli utenti per massimizzare il risparmio energetico e minimizzare i costi.
Il progetto è stato sviluppato con attenzione particolare agli edifici commerciali, che da soli compongono quasi il 50% del consumo energetico degli edifici americani (il 20% di tutto il consumo nazionale), consumo ripartito principalmente tra riscaldamento/raffreddamento, illuminazione e strumenti elettrici in generale.
In dettaglio, si vuole minimizzare i costi cercando di schedulare i meeting nelle varie stanze di un edificio commerciale nel modo più conveniente possibile (ad esempio due meeting con scelte di temperatura e numero di posti simili verranno schedulati nella stessa aula uno dopo l'altro).
L'idea principale è che gli utenti comunichino una richiesta di meeting al sistema con le svariate preferenze (ad esempio zona dell'edificio, orario, temperatura della sala, quantità di persone attese ecc) più una flessibilità a queste richieste.
TESLA quindi in maniera dinamica prova a schedulare tutte le richieste che riceve, cercando di non sforare la flessibilità massima. Se però uno spostamento di alcuni

parametri (esempio l'orario) può incrementare notevolmente il risparmio energetico, TESLA chiede all'utente se è possibile fare tali cambiamenti.

La computazione si divide in 4 aspetti principali: una schedulazione dell'energia il più efficiente possibile, l'identificazione dei meeting chiave che possono portare ad un notevole risparmio energetico richiedendo più flessibilità, comprensione delle preferenze dell'utente, comunicazione con l'utente.

Nel dettaglio TESLA prende in input le preferenze dell'utente, con cui cerca di fare una ottimizzazione per bilanciare il comfort dell'utente e l'assegnamento ottimo delle stanze, a questo punto di sua iniziativa collabora con l'utente al fine di identificare i punti modificabili dell'evento allo scopo di evitare costi inutili ed infine ottimizza il sistema in modo da ottenere il maggior risparmio energetico possibile, rispettando i vincoli imposti da ogni utente.

In particolare TESLA ha al suo interno due algoritmi: il primo che computa la schedulazione dei meeting, il secondo che si basa sulla schedulazione ottenuta e che cerca di modificare determinate richieste nel meeting allo scopo di aumentare il guadagno.

Il punto chiave è dunque la schedulazione, sono quindi stati rappresentati con:

- $r_i = <a_i, T_i, L_i, \delta_i, d_i, n_i>$, la richiesta di meeting, dove a_i è il tempo di arrivo, T_i i time slot preferiti per avviare l'evento, L_i i luoghi preferiti, d_i è la deadline in cui comunicare all'utente le informazioni, δ_i è la durata dell'evento e n_i è il numero di persone attese, tutti riferiti alla richiesta di meeting i-esima.
- $\alpha_i = < \alpha_i^T, \alpha_i^L, \alpha_i^d >$, la flessibilità della richiesta, dove α_i^T è la flessibilità del time slot del meeting i, α_i^L è la flessibilità della locazione ed α_i^d è la flessibilità della deadline.

Inoltre vengono creati dei set di richieste, in modo da raggrupparli in categorie distinte:

- $R^s(t)$ il set di richieste da schedulare al tempo t (quindi $t = d_i$ ed $t >= a_i$)
- $R^A(t)$ il set di richieste già assegnate prima del tempo t (quindi $t > d_i$ ed $t > a_i$)
- $R^K(t)$ il set di richieste arrivate al tempo t ma che verranno schedulate in futuro (quindi $t < d_i$ ed $t >= a_i$)
- $R^U(t)$ il set di richieste sconosciute future (quindi $t < d_i$ ed $t < a_i$)

Partendo da queste richieste verrà calcolato quindi lo scheduling che minimizza i costi, calcolati come il costo dell'energia elettrica per assegnare le stanze a chi ne fa richiesta. Importante precisare che l'algoritmo si divide in due stadi:

- Nel primo stadio viene calcolata una soluzione x^* tale per cui il costo *e* (che è il costo totale nello schedulare gli eventi in un luogo L al tempo T) dello stadio uno e il valore atteso del costo E dello stadio due siano minimizzati. Il primo costo *e* è semplicemente la somma dei singoli costi nello schedulare un evento al tempo T nel luogo L, mentre il costo E viene calcolato nel secondo stadio.
- Nel secondo stadio, appunto, viene calcolata la E, che è una previsione dei costi e sfrutta una funzione $Q(x, \xi^n)$, che è la funzione dei valori del consumo di energia futuro, e la probabilità p_n^U che ξ^n si realizzi. ξ è il vettore che determina le future richieste di meeting. Quindi $E = \sum_{n=1}^{N} p_n^U * Q(x, \xi^n)$.

Il secondo algoritmo come detto cerca di trovare i meeting chiave e di aumentare la loro flessibilità allo scopo di ottenere un miglioramento generale dei costi. L'algoritmo che si occupa di questo, prima prende tutti i meeting candidati e decide quali sono quelli che, se modificati, porterebbero ad un miglioramento delle prestazioni sensibile. A questo punto si potrà avere da 0 ad n utenti candidati possibili tra quelli che hanno proposto un meeting, che verranno interpellati per ottenere una flessibilità maggiore.

Il caso viene studiato in un edificio in particolare, una delle principali librerie dell'università Southern California, la quale ha al suo interno circa N = 300 meeting al giorno in circa 35 sale, lavorando per 24 ore al giorno, 7 giorni la settimana, ad eccezione delle feste nazionali.

Nel paper vengono quindi presentati alcuni dati sull'utilizzo di TESLA nell'ambiente sopra descritto e con i dati reali del luogo, in cui viene dimostrato un miglioramento di circa il 50%, con un risparmio di oltre 18 mila dollari all'anno in un singolo edificio commerciale.

Inoltre all'aumentare degli N campioni (numero di meeting) il risparmio che ne deriva dall'utilizzo di TESLA aumenta.

Un altro punto fondamentale è la disponibilità alla flessibilità da parte degli utenti. Dai campioni raccolti in questo paper è stata riscontrata una flessibilità media per il tempo del 25% circa, per la locazione del 16% circa ed un risparmio supplementare medio del

30% utilizzando un sistema di flessibilità richiesta (cioè il caso in cui all'utente si chiede maggiore flessibilità allo scopo di ottenere maggiori guadagni).
Il risultato ottenuto è dunque un sensibile miglioramento (addirittura fino al 50%) in un ambiente di tipo lavorativo e con risparmi sempre più sensibili all'aumentare dei campioni e della flessibilità dell'utenza, dimostrando che un'ottimizzazione su larga scala e la flessibilità da parte dell'utente porta ad un risparmio energetico.
In questo caso le similitudini col nostro lavoro esistono, infatti TESLA chiede una flessibilità all'utente, che può essere vista come, nel nostro caso, i diversi risultati restituiti dal livello bottom al livello top. La differenza sostanziale sta nel fatto che il livello bottom nel nostro caso fa già un'ottimizzazione sui dispositivi, mentre nel caso di TESLA l'utente richiede semplicemente la stanza per un determinato orario e per un determinato periodo, senza preoccuparsi di ottimizzarne l'utilizzo.
Un possibile ulteriore sviluppo di TESLA è presentato in [7], dove viene introdotto un nuovo agente (THINC, Tool For Human INcentivization and Cooperation) che per incentivare la flessibilità degli utenti distribuisce dei crediti che fungono da "premio" per gli utenti. Maggiore sarà il risparmio ottenuto per merito di tale flessibilità e maggiore sarà dunque il premio.
I valori da distribuire agli utenti sono calcolati tramite un algoritmo di Shapley Value approssimato che utilizza il concetto di Coalitional Game modellato come (N,v) dove:

- N è il set dei giocatori o in questo caso il set delle richieste di meeting.
- S è la coalizione che è un sottoinsieme di N
- v è la funzione caratteristica, nel nostro caso v(S) è il risparmio energetico totale calcolato come e'(S) – e(S) dove e'(S) è l'energia richiesta senza dare flessibilità, mentre e(S) al contrario è l'energia richiesta quando viene concessa una certa flessibilità.

A questo punto si calcola lo Shapley Value in modo approssimato campionando in modo casuale il valore, tramite un algoritmo chiamato ApproShapley, un meccanismo di campionamento che restituisce un valore approssimato in tempo polinomiale.
Inoltre vengono divise le richieste di meeting in set indipendenti, in quanto è dimostrato che lo Shapley Value può essere calcolato in modo separato per ogni set, aumentando la velocità di calcolo.

Dai dati raccolti si è notato che questo valore approssimato si discosta dal valore reale di circa il 10%.
Ovviamente la qualità della soluzione migliora rispetto ad altri algoritmi (come ad esempio TESLA) perché l'utente è invogliato ad aumentare la sua flessibilità per ottenere un premio più alto ed allo stesso modo la percentuale di flessibilità di tempo (27% in media) e luogo (42%) aumenta.
Valgono quindi le conclusioni illustrate per TESLA, con l'aggiunta che in [7] vi è l'introduzione di un coalitional game, che rende quindi la formalizzazione decisamente diversa dal nostro caso.

2.4) Distributed Multi-Period Optimal Power Flow for Demand Response in Microgrids [4]

L'idea di fondo di questo lavoro è quella di utilizzare un sistema ADMM (Alternating Direction Method of Multipliers) per risolvere un problema OPF (Optimal Power Flow), adattandolo in modo tale da permettere agli agenti di negoziare il consumo di energia in modo cooperativo.
La maggior parte dei lavori precedenti in questo ambito si focalizzano sull'uso di prezzi Real Time, combinando una minimizzazione dei costi con una massimizzazione del comfort, oppure usano modelli ADMM ma non modellano reti elettriche e quindi non si è in grado di dedurre a priori se tale modello sarebbe funzionale e performante in questi casi. Segue dunque formalizzazione del problema, che si divide in 3 parti fondamentali:

- Componenti: ($c \in C$) sono i carichi veri e propri. Ognuno ha dei terminali con cui può connettersi ad altri componenti. Ogni terminale del componente è modellato con un vettore $y_i = (p, q, v, \theta)$, dove p e q rappresentano la potenza reale e quella reattiva, v rappresenta il voltaggio e θ la fase.
  I componenti hanno un vettore x_c (che contiene tutti i vettori y_i di tutti i terminali del componente), una funzione di costo f_c, funzione del vettore x_c ed una funzione di vincolo g_c, anch'essa funzione di x_c, tale che $g_c(x_c) \leq 0$.
  Essi si dividono in bus, linee, generatori e carichi spostabili. Ovviamente i più interessanti sono quest'ultimi. Essi sono componenti con un singolo terminale

usati per modellare i carichi elettrici e sono rappresentati da un profilo di potenza che è pari alla potenza nominale moltiplicata per il numero di time step in cui rimane acceso l'apparecchio.

- Connessioni: ($l \in L$) una connessione connette due terminali. Ogni terminale, come abbiamo visto, è rappresentato da un vettore y. In una connessione tra due terminali le somme delle due p e delle due q e le sottrazioni delle due v e delle due θ saranno pari a 0 (in accordo con la legge di Kirchhoff).
- Problema di ottimizzazione: non è altro che la minimizzazione della somma delle funzioni di costo f_c di tutti gli apparecchi presenti nella rete.

Come detto l'algoritmo sarà però distribuito e per ottenere questo verrà usato il modello ADMM che è una variante del metodo dei moltiplicatori di Lagrange.

L'algoritmo nel dettaglio usa due set di componenti (C_1, C_2) e due vettori di variabili (x_1, x_2), una variabile k che indica il numero di iterazione ed i valori y_i e $\lambda_{i,j}$ inizializzati ad un certo valore noto. Per l'iterazione k qualunque l'algoritmo procede in questo modo:

- Ottimizza L (funzione di Lagrange) su x_1, mantenendo costante x_2 al suo valore k-1
- Ottimizza L (funzione di Lagrange) su x_2, mantenendo costante x_1 al suo valore k
- Aggiorna la variabile $\lambda_{i,j}$

Quindi una singola iterazione corrisponde ad una doppia fase di ottimizzazione. Nei diversi test è dimostrato che il problema ADMM trattato converge all'ottimo globale con una variazione massima dell'1% (variabile in base ai modelli di potenza utilizzati) e con un tempo che varia da pochi secondi fino ad un massimo di un paio di minuti (anche in questo caso, differente in base ai modelli di potenza utilizzati).

Il lavoro riesce quindi a dimostrare che l'uso di un sistema distribuito allo scopo di ottenere un assegnamento ottimo delle risorse nel tempo (scheduling) in una rete elettrica non solo è possibile ma produce anche dei buoni risultati, con tempi più o meno accettabili in base al modello di potenza.

Pur risolvendo un problema simile a quello che vogliamo tentare di risolvere in questa tesi, tale articolo utilizza una formulazione totalmente differente, in quanto sfrutta la

possibilità di spostare i carichi temporalmente ma cerca un ottimo tramite una formulazione ADMM e sfruttando il metodo dei moltiplicatori di Lagrange. Inoltre anche in questo caso non è presente la divisione in livelli.

2.5) Managing Power Flows in Microgrids Using Multi-Agent Reinforcement Learning [5]

L'idea alla base di questo paper è un po' diversa da quelli visti fino a qui. L'ambiente, similmente a quanto visto fino a qui, è quello delle reti elettriche attuali, che sono sicuramente molto espandibili e proprio per questo in rapida crescita, con danni sempre maggiori per l'ambiente.

In questo articolo si vuole tentare di utilizzare dei sistemi di accumulo intelligenti (batterie), con cui è possibile ottenere vantaggi dalle energie rinnovabili, caricando o scaricando in modo opportuno tali apparecchi. L'idea è ovviamente quella di caricare le batterie quando vi è un eccesso di potenza generata dagli apparecchi (ad esempio pannelli solari o pale eoliche) e al contrario scaricarle nel momento in cui questa fornitura viene a mancare.

Come al solito viene usato un approccio di tipo MAS (cioè un sistema multi agente) mentre la risoluzione del problema viene eseguita tramite Reinforcement Learning (RL). Questi tipi di sistemi apprendono interagendo con l'ambiente in cui sono immersi: ad ogni time step l'agente percepisce l'ambiente ed esegue un'azione, che causa una transizione in un nuovo stato all'ambiente. Dei premi, che hanno un valore scalare, vengono assegnati a tale transizione in base alla qualità dello stato raggiunto e col tempo il sistema impara a massimizzare tali ricompense. Una risoluzione RL utilizzata in un sistema MAS è detta MARL.

In Figura 2.2 è rappresentata in modo simbolico la rete.

L'articolo passa quindi ad una rappresentazione della stessa:

- I_{dc} è la corrente presente nel bus (rappresentato in figura) ed è calcolata come la capacità C negativa, moltiplicata la derivata rispetto al tempo di V_{dc}, quindi:
  $I_{dc}(t) = -C^* (dV_{dc}(t)/dt)$

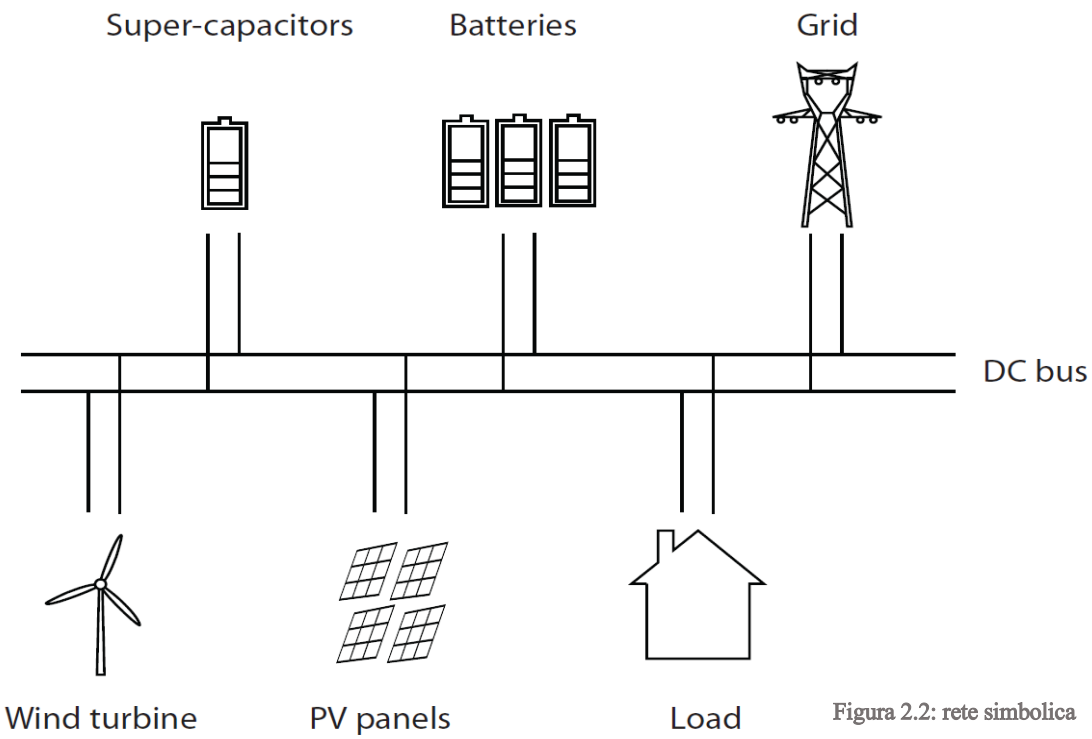


Figura 2.2: rete simbolica

- La V_{dc} sopra introdotta dovrà essere compresa in un range di valori ($V_{dc}^{min} \leq V_{dc}(t) \leq V_{dc}^{max}$) ed è calcolata come la V_{dc} iniziale sottratta ad 1/C moltiplicato con l'integrale della corrente I_{dc}: $V_{dc}(t) = V_{dc}(0) - \frac{1}{C} \int I_{dc}(t)dt.$
- In accordo con la legge di Kirchhoff, le somme delle correnti in un nodo della rete deve essere pari a zero.
- Sono inoltre rappresentate la potenza dei diversi apparati di accumulo: $P_{sto;k}(t) = \frac{I_{sto,k} * V_{dc}}{n_k}$ se $I_{sto,k} \geq 0$, altrimenti : $P_{sto;k}(t) = I_{sto,k} * V_{dc} * n_k$, dove n_k è l'efficienza di conversione.

  Ed il loro stato di carica: $SOC_k(t) = SOC_k^{init} - \frac{\int P_{sto,k}(t)\, dt}{E_k}$, dove SOC_k^{init} è lo stato di carica iniziale ed E^k è la capacità dell'apparato di accumulo.

L'ottimo consiste nel decidere, ad ogni istante di tempo t, per ogni sistema di accumulo e per la smart grid, quanta corrente bisogna sviluppare, in modo che V_{dc} resti stabile.

A questo punto si può modellare il sistema come un RL. Per farlo si sfruttano le catene di Markov, infatti il processo può essere visto come un problema di decisione di Markov M=(X, U, f, p, T, v), dove X è il set degli stati percepibili, U sono le possibili azioni, f

è la funzione di transizione di stato, p è la funzione che calcola il premio per una transizione, T è l'orizzonte temporale, v è il fattore di sconto.

Se f è incognita (come nel nostro caso), allora si sfrutta il Q-learning, che ricava un valore Q (che rappresenta il premio di uno stato eseguendo una determinata azione) tramite il quale si può ottenere h*(x) che è l'azione ottima per ogni stato x.

Si sfrutta dunque l'algoritmo MARL per il calcolo del valore di Q tramite approssimazioni, che viene sfruttato per calcolare h*(x) come il massimo delle Q stesse (diverse in base alla combinazione stato/azione).

I vantaggi più rilevanti di questo approccio sono che l'algoritmo è totalmente distribuito e un nuovo apparecchio può entrare immediatamente in funzione.

I risultati ottenuti sono interessanti e le prove sono state svolte in 16 diversi scenari e con un numero diverso di batterie e condensatori (al massimo 2 per tipo), da cui è subito chiaro che le batterie possono dare meno energia ma per più tempo, mentre i condensatori possono fornire una grossa quantità di energia ma per un brevissimo tempo. Allo stesso modo è stato dimostrato come un uso di un numero maggiore di strumenti uguali (ad esempio 2 batterie) migliori sensibilmente i risultati. Dalle prove effettuate risulta dunque una diminuzione effettiva della richiesta energetica ai fornitori, sfruttando l'energia presente nelle batterie, che però non fa in tempo a rigenerarsi nel caso in cui venga utilizzata da sola (al contrario di quanto avviene se usata in combinazione con un'altra batteria). Come accennato invece un uso dei condensatori (Super-capacitors nella figura) da soli sarebbe impossibile perché questi danno molta energia per un tempo brevissimo e quindi dopo un primo istante in cui non viene richiesta energia ai provider, si avrebbe per il resto del tempo invece una richiesta continua a quest'ultimi.

Questo progetto potrebbe essere interessante nel nostro caso per un eventuale sviluppo futuro, che tiene in considerazione anche eventuali risorse rinnovabili presenti nel complesso, come ad esempio pannelli solari.

2.6) Optimizing the Energy Exchange between the Smart Grid and Building Systems [6]

In questo paper, speciale attenzione è data all'ottimizzazione dello scambio di informazioni ed energia tra sistemi per gestire l'energia in edifici (BEMS) e le Smart Grid (SG).

Matematicamente il sistema è molto complesso perché soggetto a diverse variazioni anche impreviste, come ad esempio variazioni climatiche, tipi e metodi di costruzione dell'edificio, dalle proprietà termiche dei materiali usati, dagli occupanti e dalle loro preferenze di comfort, ecc.

Dato che sviluppare una soluzione unica per ottimizzare un tale sistema, con le diverse interazioni tra reti ed edifici, è praticamente impossibile, gli autori hanno deciso di dividere il sistema in 3 parti:

- Il livello "edificio".
- Il livello vicinato (una sorta di rete tra edifici vicini).
- Il livello SG.

Ovviamente come visto fino a qui, il sistema si avvale di un approccio MAS.

La Figura 2.3 rappresenta il funzionamento concettuale del sistema.

I profili blu rappresentano gli agenti a livello "edificio", mentre quelli neri modellano il livello SG

Il paper quindi propone un approccio per modellare l'ottimizzazione in un sistema con interazioni SG-BEMS, ed in particolare si offrono due possibili strade: quella della teoria dei giochi dinamica (rappresentata dal primo modello) e quella dell'ottimizzazione stocastica (rappresentata dalla modellazione secondo Markov).

Il primo modello consiste in una tupla <N, X, q, v> dove:

- N è il set degli n agenti.
- X è il set dei m discreti e consecutivi time slot.
- $q = (q_1 \dots q_m)$ è il vettore dei prezzi massini che si è disposti a pagare per l'energia elettrica. Quindi q_j non è altro che il prezzo massimo al time slot x_j.
- $v = (v_1 \dots v_n)$ è il vettore delle funzioni di valutazione, dove v_i è la funzione di valutazione dell'agente i. Questa è una funzione sulle possibili allocazioni dei carichi nel tempo, parametrizzata da due valori: d_i, la deadline dell'agente i ed λ_i, il numero di time slot richiesti dall'agente i.

Il valore di v_i è pari a w_i se nell'allocazione F_i avrò λ_i time slot temporalmente prima di d_i (o in altre parole se l'allocazione dell'agente gli lascia il tempo per terminare la computazione), altrimenti il valore sarà pari a 0.

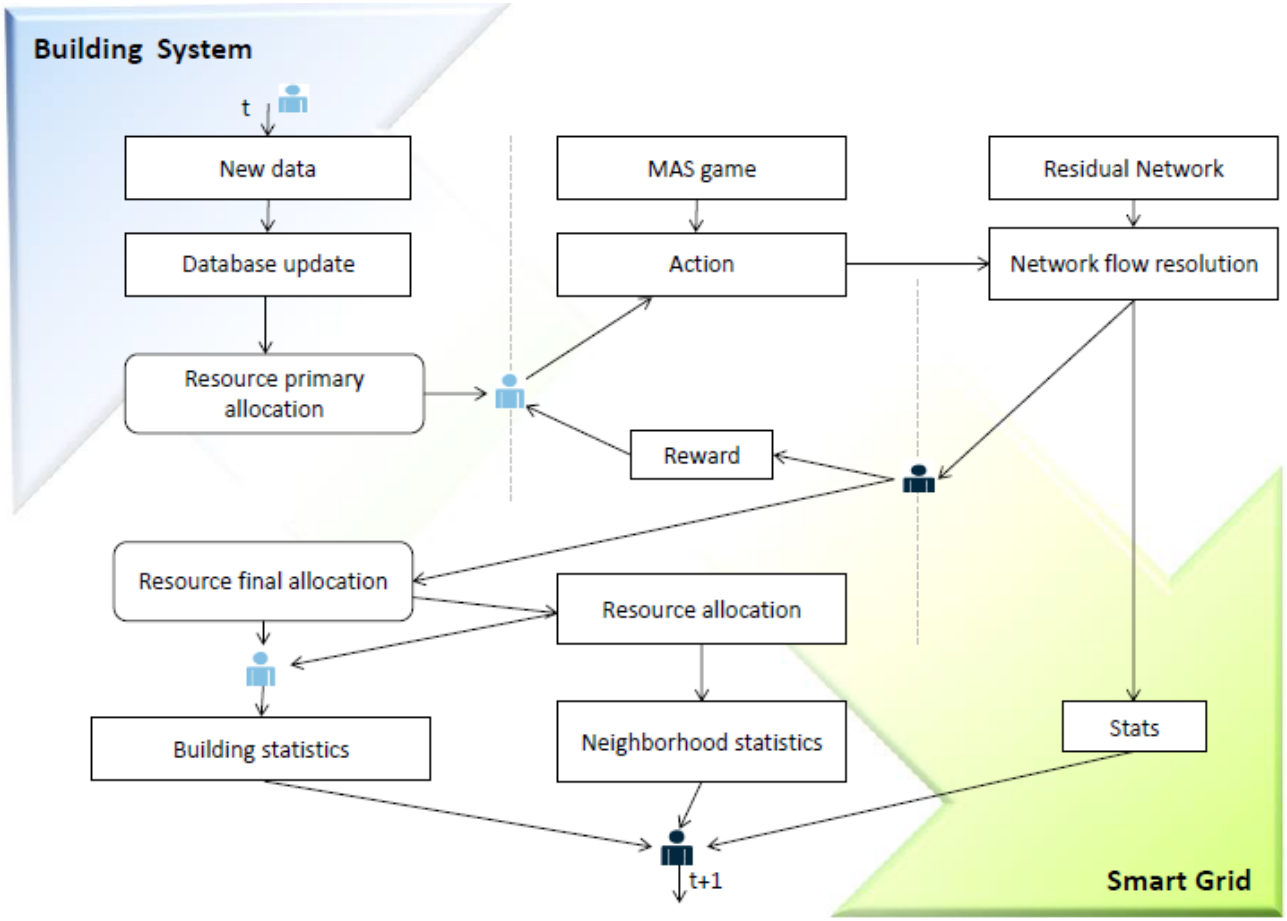


Figura 2.3: funzionamento del sistema

L'allocazione ottima F sarà data dalla massimizzazione tra la somma dei prezzi q_j con i valori w_i (che chiameremo V), sottratti con la somma dei prezzi competitivi di equilibrio (Q):

$F = argmax(V - Q)$.

Per cercare di semplificare tale algoritmo si può dividere i time slot X in subset S e fare valutazioni in questo bundle ristretto.

Il secondo metodo utilizza invece le catene di Markov, che sono modellate da una tupla <Q, N, A, P, r> dove:

- Q è l'insieme finito di possibilità (ad esempio dei diversi assegnamenti). In altre parole sono i diversi modi in cui il problema può essere risolto;
- N è l'insieme finito di agenti;

- A è l'insieme finito di possibili azioni per l'agente i;
- P è la probabilità di transizione partendo dallo stato q e andando allo stato q', con una azione a.
- r è il premio per l'agente i e dipende dallo stato q e dall'azione a.

L'allocazione ottima sarà tale da massimizzare il premio per ogni agente (e risolto coi metodi classici delle catene di Markov).

Si nota come tale lavoro sia simile a quanto si intende sviluppare in questo lavoro di tesi, in quanto è presente la divisione in diversi livelli.

La differenza più grossa, oltre dagli strumenti utilizzati, deriva dal modo in cui il problema viene modellato, con teoria dei giochi e catene di Markov in questo lavoro, come problemi DCOP risolti tramite formalizzazione per il sistema FRODO nel nostro caso.

2.7) Demand response for home energy management system [8]

In questo paper viene propost un algoritmo DR (Domanda-Risposta) che controlla i diversi sistemi elettrici di un'abitazione in modo centralizzato. Questo programma potrà funzionare in modo accettabile anche con risorse che hanno capacità di calcolo limitate, come ad esempio dei contatori intelligenti.

Nella Figura 2.4 viene schematizzato un esempio di algoritmo DR in un sistema casalingo, dove gli elettrodomestici sono collegati in rete tra loro e controllati da un sistema di gestione (HEMS, home energy management system).

Prima di tutto, viene formulato un problema non lineare con una ottimizzazione mista discreto/continuo. Alcuni software commerciali come CPLEX o algoritmi euristici come PSO (Particle Swarm Optimization) e greedy possono essere usati per risolvere questi problemi. Però tali sistemi hanno esigenze di calcolo importanti, con convergenza al risultato lenta e costi di computazione altrettanto importanti e che non sono adeguati alla potenza di elaborazione fornita da semplici sistemi come contatori intelligenti.

Per questo motivo, in questo paper, è proposto un metodo dei gradienti basato su un algoritmo PSO (GBPSO).

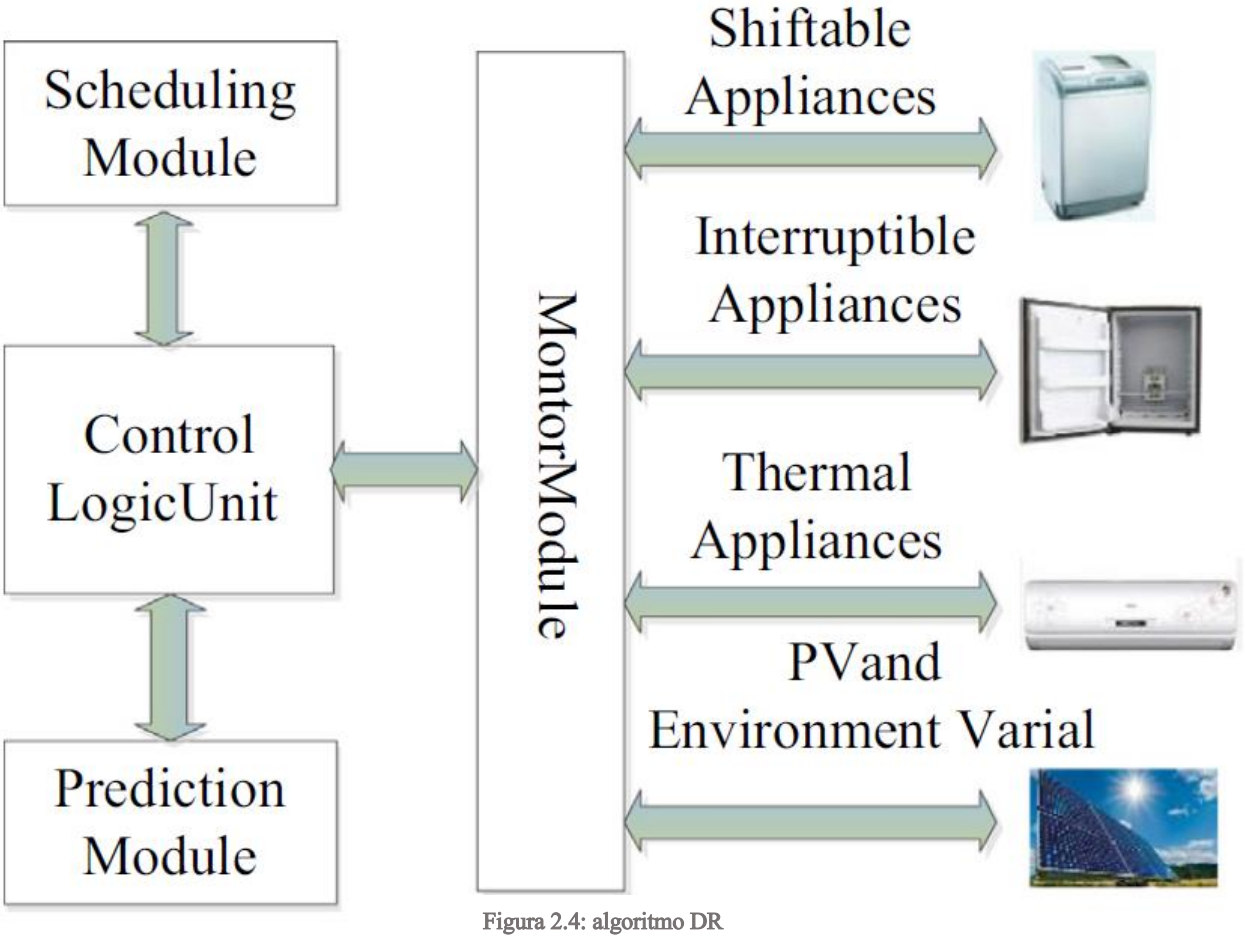


Figura 2.4: algoritmo DR

La rete è costituita da un generatore fotovoltaico, dei dispositivi di stoccaggio, dei carichi elettrici differibili e dei carichi termici.

Quindi il programma DR, definito come un problema di ottimizzazione, calcola una schedulazione dell'accensione dei carichi e la logica di controllo manda tale schedulazione ai diversi carichi.

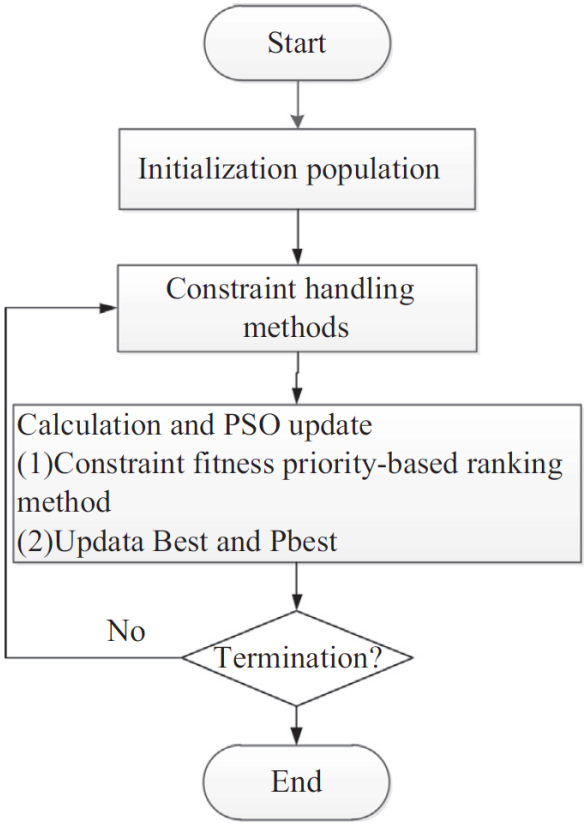
L'obiettivo è quindi di determinare i migliori orari di funzionamento per gli apparecchi differibili e per gli apparecchi termici e la quantità di carica delle batterie per ogni passo temporale, in modo da minimizzare il costo dell'elettricità considerando i limiti fisici della rete e degli apparecchi:

- Viene definita $P_{grid}(h)$ come la potenza comprata dall'utilizzatore se tale valore è positivo, mentre è la potenza venduta se tale valore è negativo al tempo h. P_{grid} è la somma di tutte le potenze dei diversi dispositivi (batteria, carichi differibili e non ecc) presenti nella rete.
- Vengono definiti dei constraint per le batterie come visto nella Sezione 1.5 (vincoli di potenza massima e minima e vincoli di stato di carica).

- Inoltre si costruiscono dei vincoli per le temperature nell'ambiente ($T_{in}^{min} \leq T_{in} \leq T_{in}^{max}$). $T_{in}(h)$ è calcolato in modo simile alla Sezione 1.1.
- $Min\ Cost = \sum_{h=1}^{24}[TOU(h) * P_{grid}(h)]$, dove TOU è la tariffa al momento dell'uso ed è noto.

Nella Figura 2.5 è rappresentato il funzionamento del programma DR in modo schematico.

Figura 2.5 funzionamento programma DR



1. **Initialization population**: vengono generati N numeri casuali iniziali che rappresentano la popolazione iniziale. Ognuno di essi verrà chiamato "particella". Ognuna di queste particelle genera a sua volta un vettore X_{dis} che contiene $X_{battery}[1\ldots\ 24]$, $X_{def}[1..24]$ ecc, che rappresentano gli orari di carica/scarica delle batterie, gli orari di accensione dei carichi differibili ecc.
2. **Constraint handling methods:** è il metodo che si occupa di quelle particelle che violano i constraint. Allo scopo viene utilizzato il metodo gradient-based repair così schematizzato:

- Step1: per ogni soluzione controllare il numero di vincoli violati. Se la soluzione non è possibile (cioè ha un numero di vincoli violati maggiore di 0) allora si salta allo step2, altrimenti si termina.
- Step 2: si calcolano $\nabla C(x)^{-1}$ e Δx.

  $\nabla C(x)^{-1}$ è l'inverso della matrice dei gradienti di C(x), C(x) è il vettore dei vincoli. In particolare $\nabla C(x)$ è calcolata come:

$$\nabla C(x) = \begin{pmatrix} \frac{\partial g_1(x)}{\partial x_1} & \frac{\partial g_1(x)}{\partial x_2} & \cdots & \frac{\partial g_1(x)}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial g_q(x)}{\partial x_1} & \frac{\partial g_q(x)}{\partial x_2} & \cdots & \frac{\partial g_q(x)}{\partial x_n} \\ \frac{\partial h_{q+1}(x)}{\partial x_1} & \frac{\partial h_{q+1}(x)}{\partial x_2} & \cdots & \frac{\partial g_{q+1}(x)}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial h_m(x)}{\partial x_1} & \frac{\partial h_m(x)}{\partial x_2} & \cdots & \frac{\partial h_m(x)}{\partial x_n} \end{pmatrix}$$

  Dove g ed h sono i vincoli, che fanno appunto parte del vettore C(x), dove $C(x) = (g_1(x) \ldots g_q(x)\ h_{q+1}(x) \ldots h_m(x))^t$.

  Δx invece è calcolata come il prodotto tra $-\nabla C(x)^{-1}$ e $\Delta C(x)$, dove $\Delta C(x)=(\Delta g_1(x) \ldots \Delta g_q(x)\ h_{q+1}(x) \ldots h_m(x))^t$. Δx rappresenta l'incremento del punto x per soddisfare i vincoli.
- Step3: si calcola il nuovo vettore soluzione come $x^{new} = x + \Delta x$.
- Step4: si valuta la nuova soluzione, se è ancora impossibile si salta allo step 2, altrimenti si termina.

3. **Calculation and PSO update:** vengono quindi ordinate le particelle in base al loro valore di utilità, calcolato sulla base dei constraint. Sfruttando quindi il metodo PSO per la creazione della nuova popolazione e comparandola con la vecchia vengono selezionate le particelle migliori.

4. **Termination:** l'algoritmo PSO termina quando si eccede il numero massimo di generazione di nuove popolazioni oppure quando una soluzione migliore a quella attuale non può essere trovata entro un certo range di iterazioni.

Nel paper è poi presente una comparazione delle performaces tra questo metodo ed un metodo commerciale come CPLEX, HPSO (Hybrid PSO) ed COPSO (COoperative PSO), tutti con una popolazione di 200 particelle ed eseguiti 10 volte ognuno. Nella Figura 2.6 sono indicati e confrontati i risultati.

Figura 2.6: Risultati degli esperimenti

	Best solution	Average solution	Cpu time (s)	Feasible solution	Cpu time (s)
CPLEX [11]	18.8861	18.8861	0.015	18.8861	0.015
Gradient based PSO	19.8854	20.1567	4.5015	21.8602	0.093
COPSO [21]	22.7191	23.9797	9.628	26.1069	2.934
HPSO [16]	23.005	24.4585	26.4433	26.6892	2.64

Come mostrato in tabella, sia CPLEX, che il metodo di questo paper restituiscono una soluzione possibile in meno di 0.1 secondi (cosa molto utile in quanto si sta lavorando con sistemi DR).

Inoltre il costo temporale per il calcolo della soluzione ottima è maggiore in GBPSO, ma comunque tale algoritmo converge velocemente ad un ottimo.

Gli altri due algoritmi invece sono visibilmente peggiori, sia in termine di tempo che di qualità della soluzione.

Quindi si nota che tra gli algoritmi euristici GBPSO è molto performante, avvicinandosi all'algoritmo commerciale CPLEX.

Come detto all'inizio del capitolo, questo è quindi un possibile algoritmo centralizzato per risolvere i problemi di ottimizzazione (applicabile ad esempio a livello top).

3 – FRODO

In questo capitolo verrà presentato il framework FRODO, utilizzato per risolvere i problemi DCOP presentati in questa tesi (si evince quindi che la formulazione finale dovrà essere implementata con uno standard coerente a quello di FRODO).
FRODO (FRamework for Open/Distributed constraint Optimization, raggiungibile al seguente link: http://frodo2.sourceforge.net/) è un framework Java open-source per la soluzione di problemi di ottimizzazione vincolata distribuita (DCOP), sviluppato inizialmente all'Artificial Intelligence Laboratory (LIA) dell'Ecole Polytechnique Fédérale de Lausanne (EPFL), in Svizzera.
Per la realizzazione di questa tesi abbiamo utilizzato la versione 2.12.1 di FRODO: tale framework si occupa di simulare in una singola macchina virtuale Java un ambiente multiagente, in cui ogni agente è eseguito in maniera asincrona in un thread dedicato e comunica con gli altri attraverso scambi di messaggi.

3.1 – FRODO, architettura

FRODO è strutturato con un'architettura multistrato modulare.
Lo strato di comunicazione, responsabile dello scambio di informazioni fra gli agenti, è implementato come una coda di messaggi Java. Tipicamente ogni agente possiede una coda, che utilizza per ricevere e inviare messaggi.
Il funzionamento a livello logico è il seguente: viene dato uno spazio di possibili assegnamenti ad alcune variabili definite, e lo spazio delle soluzioni è rappresentato dai possibili assegnamenti che corrispondono alle soluzioni del problema dato. Questo è lo strato degli spazi delle soluzioni. Essi sono associati sempre ad una utilità (o a un costo) che misura la qualità di ogni soluzione.
Ad esempio una variabile X_1 può avere come dominio {100,150 e 200} mentre X_2 {150, 170}, gli assegnamenti possibili saranno ad esempio {100,150}, {150,170} ed {200,150}.
Gli assegnamenti che sono invece inammissibili possono essere rappresentate da un costo fittizio di peso infinito.

Gli spazi delle soluzioni possono essere espressi anche in maniera intensionale, basandosi sui vincoli JaCoP.

Lo strato degli algoritmi, infine, si occupa dell'implementazione vera e propria degli algoritmi distribuiti utilizzati per risolvere i DCOP.

Ogni algoritmo è implementato come uno o più moduli, i quali descrivono come gli agenti devono comportarsi alla ricezione dei vari messaggi e quali messaggi devono inviare agli altri agenti o agli altri moduli. Questa struttura modulare rende possibile un alto grado di personalizzazione degli algoritmi, oltre che il riutilizzo e la manutenzione del codice.

Fra gli algoritmi supportati da FRODO, abbiamo utilizzato i seguenti:

- DPOP: (Dynamic Programming Optimization Protocol) è un algoritmo completo e corretto (trova quindi la soluzione ottima se esiste). L'algoritmo si divide in tre fasi: la costruzione di uno pseudo-albero (dovuto al fatto che DPOP deriva da algoritmi che usano reti di alberi per la risoluzione delle reti), la propagazione delle utilità e la propagazione dei valori.

  Durante la fase di propagazione i figli degli alberi mandano ai propri padri i messaggi UTIL fino a che si arriva al nodo radice.

  A questo punto il nodo radice assegna alla propria variabile il valore che massimizza l'utilità del proprio sottoalbero e invia ai figli un messaggio VALUE, tramite il quale i figli possono scegliere il loro valore (che massimizza il loro sottoalbero), fino a raggiungere le foglie dell'albero.

  Il punto di forza di DPOP riguarda la complessità in termini di messaggi scambiati, che cresce linearmente rispetto all'altezza dello pseudo-albero, generando quindi un basso overhead di comunicazione. Tuttavia le performance possono essere influenzate dalla dimensione dei messaggi UTIL, che cresce esponenzialmente rispetto alla larghezza dello pseudo-albero.
- ADOPT: (Asynchronous Distributed OPTimization) è un algoritmo corretto (ossia termina sempre producendo un risultato corretto) e completo, in grado di trovare una soluzione ottimale al problema o almeno una soluzione a una distanza massima dall'ottimo definita dall'utente.

  L'idea principale dietro ADOPT è di ottenere asincronismo permettendo a ogni agente di cambiare il valore assegnato alla propria variabile quando scorge la

possibilità di trovare una soluzione migliore rispetto a quella attuale. Questa strategia di ricerca permette asincronismo perché un agente non necessita una conoscenza globale del problema per prendere una decisione riguardo al suo problema locale.

Dato che questa strategia permette di abbandonare soluzioni prima che sia provata la loro ottimalità locale, è possibile che alcune di esse siano riutilizzate.

La seconda idea chiave in ADOPT è quindi quella di ricostruire in maniera efficiente soluzioni considerate precedentemente (con complessità spaziale polinomiale) attraverso l'uso di una soglia di backtrack, cioè una tolleranza sul costo della soluzione che previene l'utilizzo di backtracking indiscriminato.

La terza caratteristica principale di ADOPT consiste nel fornire un meccanismo di individuazione della terminazione all'interno dell'algoritmo stesso: gli agenti terminano infatti quando trovano una soluzione completa il cui costo è al di sotto della loro soglia di backtrack.

Gli agenti scambiano quindi tra loro messaggi di VALUE (da un nodo verso i figli), COST (da un nodo verso il padre, rappresenta il costo della scelta comunicata tramite VALUE) e THRESHOLD (da un nodo verso i figli, rappresenta la soglia di backtrack).

Ogni agente mantiene traccia degli assegnamenti dei vicini con priorità maggiore attraverso un contesto, ovvero una soluzione parziale contenente gli assegnamenti noti che viene cambiato ogni volta che si riceve un messaggio VALUE o se si decide di cambiare il valore della propria variabile dopo la ricezione di un messaggio COST.

L'algoritmo giunge al termine quando i limiti inferiore e superiore dell'agente al nodo radice sono uguali, il che significa che è stata trovata la soluzione ottima.

Lo svantaggio principale di ADOPT è la sua complessità in termini di messaggi scambiati, che cresce esponenzialmente rispetto al numero degli agenti: le performances degradano esponenzialmente al crescere di essi.

- MGM: MGM (Maximum Gain Messages) [15] è l'unico algoritmo incompleto che abbiamo testato, in quanto non garantisce di trovare la soluzione ottima, se esiste. MGM prende spunto dalla teoria dei giochi, in quanto i DCOP vengono modellati come giochi in cui i giocatori sono le variabili controllate dagli agenti,

le azioni sono gli assegnamenti di valori, e le informazioni note sono i valori noti degli agenti vicini. In questo modo, una soluzione corrisponde a un equilibrio di Nash del problema rappresentato.

L'algoritmo agisce in diversi turni (round): a ogni round gli agenti calcolano la propria utilità locale. Poi, gli agenti mandano un messaggio a tutti i vicini, indicando se l'assegnamento sortirà un guadagno o no. Un assegnamento è eseguito solo dall'agente che ne trarrà il maggior guadagno. L'algoritmo termina se non esiste nessun ulteriore assegnamento che può generare un guadagno.

L'aspetto più importante di questo algoritmo è la monotonicità: qualsiasi risultato ottenuto ad un round qualsiasi non sarà sicuramente peggiore di quello ottenuto ad un round precedente.

3.2 – FRODO, formato di input.

FRODO richiede in input due tipi di file: i file che definiscono il problema di ottimizzazione da risolvere e i file di configurazione che definiscono l'algoritmo da utilizzare.

Il formato del file utilizzato per descrivere i DCOP è basato su XCSP 2.1, un formato utilizzato per rappresentare reti di vincoli tramite XML, con delle piccole estensioni necessarie a renderlo adatto ai problemi di ottimizzazione distribuita.

Questo file XML è composto da 5 sezioni principali:

- La sezione <agents> in cui si definiscono gli agenti del DCOP.
- La sezione <domains> definisce il dominio dei valori delle variabili del DCOP.
- La sezione <variables> definisce le variabili del DCOP, con i corrispettivi domini e agenti a cui appartengono. Ovviamente ogni agente può avere più di una variabile.
- La sezione <relations> definisce generiche relazioni sulle variabili. Una relazione ha una corrispondenza logica con i vincoli simile a quella che esiste tra i domini e le variabili: essa descrive una nozione generica su un certo numero di variabili, senza specificarne il nome; questa nozione può essere poi implementata come un vincolo, specificando le variabili coinvolte.

Sono supportate relazioni di tipo soft, che elencano tutti i possibili valori di utilità (o di costo) e i relativi valori delle variabili che sono associati a tale utilità (o costo). Ciò nonostante è possibile dichiarare delle relazioni hard usando il valore speciale infinity.

- La sezione <constraints> elenca i vincoli veri e propri del DCOP, riferendosi alle relazioni precedentemente definite, e applicandole a variabili specifiche elencate nell'attributo scope

In Figura 3.1 un esempio di file XCSP valido per FRODO.

```
<instance>
  <presentation name="sampleProblem" maxConstraintArity="2"
                maximize="false" format="XCSP 2.1_FRODO" />

  <agents nbAgents="3">
    <agent name="agentX" />
    <agent name="agentY" />
    <agent name="agentZ" />
  </agents>

  <domains nbDomains="1">
    <domain name="three_colors" nbValues="3">1..3</domain>
  </domains>

  <variables nbVariables="3">
    <variable name="X" domain="three_colors" agent="agentX" />
    <variable name="Y" domain="three_colors" agent="agentY" />
    <variable name="Z" domain="three_colors" agent="agentZ" />
  </variables>

  <relations nbRelations="1">
    <relation name="NEQ" arity="2" nbTuples="3" semantics="soft" defaultCost="0">
      infinity: 1 1|2 2|3 3
    </relation>
  </relations>

  <constraints nbConstraints="3">
    <constraint name="X_and_Y_have_different_colors" arity="2" scope="X Y" reference="NEQ" />
    <constraint name="X_and_Z_have_different_colors" arity="2" scope="X Z" reference="NEQ" />
    <constraint name="Y_and_Z_have_different_colors" arity="2" scope="Y Z" reference="NEQ" />
  </constraints>
</instance>
```

Figura 3.1: file XCSP valido per FRODO

FRODO necessita in input anche di un file di configurazione degli agenti che definisce l'algoritmo da utilizzare. In Figura 3.2 un esempio.

```
<agentDescription className = "frodo2.algorithms.SingleQueueAgent"
  measureTime = "true" measureMsgs = "false" >

  <parser parserClass = "frodo2.algorithms.XCSPparser"
          displayGraph = "false"
          utilClass = "frodo2.solutionSpaces.AddableInteger"
          countNCCCs = "false" />

  <modules>
    <module className = "frodo2.algorithms.varOrdering.dfs.DFSgenerationParallel"
      reportStats = "true" >
      <rootElectionHeuristic
        className = "frodo2.algorithms.heuristics.ScoringHeuristicWithTiebreaker" >
        <heuristic1
          className = "frodo2.algorithms.heuristics.MostConnectedHeuristic" />
        <heuristic2
          className = "frodo2.algorithms.heuristics.ScoringHeuristicWithTiebreaker" >
          <heuristic1
            className = "frodo2.algorithms.heuristics.SmallestDomainHeuristic" />
          <heuristic2
            className = "frodo2.algorithms.heuristics.VarNameHeuristic" />
        </heuristic2>
      </rootElectionHeuristic>
      <dfsGeneration className = "frodo2.algorithms.varOrdering.dfs.DFSgeneration" >
        <dfsHeuristic className =
          "frodo2.algorithms.varOrdering.dfs.DFSgeneration$ScoreBroadcastingHeuristic">
          <scoringHeuristic
            className = "frodo2.algorithms.heuristics.ScoringHeuristicWithTiebreaker" >
            <heuristic1
              className = "frodo2.algorithms.heuristics.MostConnectedHeuristic" />
            <heuristic2
              className = "frodo2.algorithms.heuristics.SmallestDomainHeuristic" />
          </scoringHeuristic>
        </dfsHeuristic>
      </dfsGeneration>
    </module>

    <module className = "frodo2.algorithms.dpop.UTILpropagation"
      reportStats = "true" />

    <module className = "frodo2.algorithms.dpop.VALUEpropagation"
      reportStats = "true" />
  </modules>

</agentDescription>
```

Figura 3.2: esempio di file di configurazione FRODO corrispondente all'algoritmo DPOP

Sono supportate le seguenti metriche di valutazione delle prestazioni:

- Numero e dimensione dei messaggi inviati: FRODO riporta il numero totale dei messaggi inviati ordinati per tipo, così come lo spazio di memoria richiesto dal messaggio più grande, e la somma totale dello spazio richiesto dalla somma di tutti i messaggi inviati. L'utilizzo di questa metrica è però computazionalmente molto oneroso, poiché misurare la dimensione dei messaggi richiede serializzazione;
- Non-Cuncurrent Constraint Checks (NCCCs): ogni agente possiede un contatore dei controlli effettuati localmente sui vincoli. Ogni messaggio inviato agli altri agenti contiene questo valore. Quando un agente riceve un messaggio, aggiorna

il proprio contatore utilizzando il valore maggiore fra quello indicato dal proprio contatore e quello indicato nel messaggio ricevuto. Il valore più grande ottenuto alla terminazione dell'algoritmo è utilizzato per ottenere una misura del lavoro di ricerca concorrente. Tale misura può essere poi utilizzata per contare il numero di NCCC (questo avviene diversamente per ogni algoritmo, in quanto dipende dalle particolari operazioni che svolgono gli agenti), e incorpora lo sforzo computazionale locale di ogni agente.

- Tempo simulato: ogni agente possiede un orologio interno. Quando invia un messaggio, l'agente aggiunge a esso un timestamp che indica il momento in cui il messaggio è stato inviato secondo il proprio orario. Quando riceve un messaggio, se il timestamp associato riporta un orario maggiore rispetto l'ora indicata dal proprio orologio interno, l'agente aggiorna il proprio orario in modo che corrisponda al timestamp del messaggio appena ricevuto. Quando l'algoritmo termina, il tempo di esecuzione è determinato dall'orario maggiore scelto fra quelli indicati dagli orologi interni di tutti gli agenti;
- Altre statistiche: diversi moduli di algoritmi possono riportare altre informazioni sul problema. Ad esempio, nel caso di DPOP, il modulo di generazione DFS può riportare in formato DOT l'albero DFS costruito e utilizzato dall'algoritmo, mentre il modulo di propagazione dei messaggi VALUE può riportare l'assegnamento ottimo delle variabili del DCOP e la corrispondente utilità (o costo) totale.

4 - Impostazione del problema applicativo

Come abbiamo detto, l'obiettivo che ci poniamo con questo lavoro è un'ottimizzazione di livello più alto (che chiameremo top) rispetto a quanto fatto fino ad oggi (dove l'ottimizzazione veniva compiuta su un singolo edificio o locale), in cui si ottimizzano i consumi di più abitazioni raggruppate in un unico complesso come un condominio oppure un residence.

Useremo quindi due livelli distinti: il livello bottom, che sarà il livello del singolo appartamento e il livello top, che sarà l'intero condominio (o residence).

I problemi a livello bottom saranno convenientemente modellati come DCOP, quello a livello top come un DCOP oppure un più semplice DCSP.

Per il livello bottom verrà utilizzato lo strumento già fornito in [10], che crea un ambiente in cui vengono simulate delle richieste di potenza simulando un ambiente reale, utilizzando [11], ma limitato a soli 2 carichi per giorno, allo scopo di ridurre la complessità del problema, pur non allontanandosi troppo da un possibile caso reale. Quindi [10], usando tale simulazione, creerà un file XML utilizzabile da FRODO, che rappresenta il singolo problema bottom, formalizzato come un DCSP.

FRODO eseguirà dunque l'ottimizzazione del problema DCOP creato restituendo come risultato gli istanti di tempo in cui i diversi elettrodomestici dell'appartamento si avviano (ogni elettrodomestico avrà più di una attività, cioè avrà dei cicli di avviamento che rappresentano il profilo di consumo dell'elettrodomestico, ognuno rappresentato da una potenza dissipata, da un tempo in cui si avvia e dal tempo per cui resta attivo).

Ovviamente se si utilizzasse a livello bottom il DCOP già implementato, si otterrebbe un singolo risultato per ogni appartamento e a livello top non si potrebbe far altro che prendere atto delle scelte e al massimo vietarne alcune per via di problemi di soddisfacibilità dei vincoli strutturali (ad esempio viene richiesta troppa potenza in un singolo istante di tempo).

Bisogna dunque avere più possibili soluzioni per ogni singolo problema DCOP a livello bottom, in modo tale da svolgere una seconda ottimizzazione a livello più alto per scegliere la combinazione di soluzioni migliore.

Un primo possibile approccio è quello di modificare il problema DCOP a livello bottom, in modo tale che esso restituisca non la scelta migliore ma le N (a scelta dell'utente o in

fase di progettazione del problema) migliori possibilità, in modo tale che a livello top si possa fare un'ottimizzazione su questi risultati.
L'inconveniente più grosso di questa possibile soluzione è la fattibilità: non è facile modificare gli algoritmi implementati nel framework FRODO che vorremmo sfruttare.
Una seconda possibile strada da seguire per aggirare i problemi implementativi sarebbe quella di far risolvere per N volte (numero di soluzioni possibili desiderate) il problema bottom, aggiungendo ogni volta dei vincoli tali da eliminare le soluzioni precedentemente selezionate. Il lato negativo è chiaramente la possibile esplosione del problema e dei tempi per la risoluzione, che potrebbero rendere il sistema inutilizzabile.
In questo approccio, quindi, si andrebbe a modificare il problema bottom, mantenendo invece inalterato quello a livello top, che rimane un normale DCOP o DCSP, in base alle scelte effettuate.
Una terza possibilità invece è quella di mantenere inalterato il problema a livello bottom, ma rieseguire l'ottimizzazione più volte, facendo variare un parametro (ad esempio la potenza massima erogabile a quella abitazione, o variando i tempi in cui è possibile avviare elettrodomestici, o usando profili di potenza specifici e variabili per ogni interrogazione ecc).
In questo modo per ogni problema bottom si otterranno risultati diversi a seconda dei valori dei parametri ed in seguito bisognerà ottimizzare l'intera struttura a livello top.
Il problema potrebbe però essere però il medesimo spiegato sopra, con i tempi che si espandono notevolmente a causa della risoluzione per N volte del problema.
Segue una iniziale formalizzazione delle possibilità sopra descritte.

4.1 - Formalizzazione problema bottom

Nel primo caso si ha un problema DCOP quindi di ottimizzazione della rete elettrica ma che restituisce più di un risultato, possiamo dunque formalizzarlo come segue:

- *n:* numero dei carichi connessi alla rete del singolo appartamento;
- *(a_1, a_2 … a_n):* insieme degli agenti del sistema. Ad ogni carico è associato esattamente un agente.

- P_{max}*:* rappresenta la potenza massima erogabile dalla rete del singolo appartamento.
- $(x_1, x_2, \ldots, x_n)$: insieme delle variabili controllate dagli agenti. Individuano il quanto di tempo di avvio degli *n* carichi.
- *T:* time step in cui è suddiviso l'orizzonte temporale.
- $(p_{11}, \ldots, p_{1m1}, p_{21}, \ldots, p_{2m2}, p_{n1}, \ldots, p_{nm})$*:* rappresenta i possibili profili di consumo degli *n* carichi, dove gli indici *i,j* stanno ad indicare l'elettrodomestico *i* e l'attività *j* dello stesso.
- $(D_{11}, \ldots, D_{1m1}, D_{21}, \ldots, D_{2m2}, D_{n1}, \ldots, D_{nm})$*:* insieme dei domini degli *n* carichi. Essi rappresentano i possibili quanti di avvio dell'*i*-esimo carico.
- *Priority:* si potrebbe anche pensare ad un livello di importanza del task, in cui alcuni elettrodomestici devono avere priorità nella potenza richiesta rispetto ad altri. Ad esempio si potrebbe pensare al forno, la mia necessità è che il forno parta immediatamente, mentre la lavatrice (a meno di vincoli da parte dell'utente) può partire anche dopo (o prima). In questo caso, il forno avrebbe priorità più alta rispetto alla lavatrice. Si è deciso di formalizzare ma di non implementare tale valore.
- La funzione di costo C è la somma dei singoli costi dell'energia elettrica per ogni carico. Lo scopo è minimizzare tale funzione rispettando vincoli e priorità. Alla fine dell'algoritmo si otterrà dunque un assegnamento ottimo X* delle variabili. Quindi $X^* = argmin\ C(x)$. Ovviamente a differenza dei problemi attualmente esistenti, X* non sarà composta da un singolo risultato ma da un vettore con N risultati, che in questa fase supponiamo essere i primi N valori ottimi. Ogni cella del vettore sarà un profilo di carico lungo quanto l'orizzonte temporale.

Il problema sarà sottoposto a vincoli, ad esempio il fatto che un elettrodomestico dovrà finire la sua esecuzione prima della fine dell'orizzonte temporale, rispetto dei vincoli di potenza massima, più possibili vincoli stabilità dall'utente.

Come spiegato sopra abbiamo due diversi modi di agire, o modificando gli algoritmi di FRODO forzandoli a restituire gli N risultati migliori alla prima esecuzione (non è però chiara la fattibilità di questa opzione), oppure risolvendo più volte lo stesso problema,

ma aggiungendo di volta in volta vincoli che escludono le soluzioni già ricavate (il problema potrebbe però esplodere).
Nel terzo caso invece, a livello bottom si avrà un problema DCOP che restituirà un singolo valore. La formalizzazione sarà dunque la medesima, ma l'assegnamento X* avrà un solo risultato, che è l'ottimizzazione effettuata. Tale caso verrà però eseguito N volte (numero di soluzioni ottime desiderate), facendo variare determinati parametri scelti a priori (il problema potrebbe però esplodere).

4.2 - Formalizzazione problema top

Nel caso in cui il problema bottom restituisca direttamente più valori allora il problema top potrà essere formalizzato nei modi seguenti.
Una prima possibilità sarà quella di utilizzare gli standard di un problema DCSP:

- *n:* numero di insiemi di carichi, ogni *n* rappresenta dunque l'intero appartamento.
- *(a_1, a_2 ... a_n):* insieme degli agenti del sistema. Ad ogni abitazione è associato esattamente un agente.
- *(x_1, x_2, ..., x_n)*: insieme delle variabili controllate dagli agenti. Individuano la scelta della soluzione per l'*i*-esimo appartamento, tra quelle messe a disposizione dal livello bottom.
- P_{max}*:* rappresenta la potenza massima erogabile dalla rete, in questo caso si intende l'intero edificio.
- *T:* time step in cui è suddiviso l'orizzonte temporale.
- *(D_{11}, ..., D_{1m1} , D_{21}, ..., D_{2m2}, D_{n1}, ..., D_{nm}):* insieme dei domini dei profili di consumo istantanei degli *n* appartamenti. Essi rappresentano le possibili scelte disponibili per ogni appartamento *i*, in accordo con i risultati ottenuti dal livello bottom.
- *Priority:* si potrebbe anche pensare ad un livello di importanza del task, in cui alcuni appartamenti devono avere priorità nella potenza richiesta rispetto ad altri. Ad esempio si potrebbe pensare ad un condominio con al primo piano dei negozi o meglio ancora uno studio (ad esempio dentistico o veterinario). In questi casi,

tali locali avrebbero una priorità più alta rispetto alle semplici abitazioni. Non è stata implementata tale opzione.

- In questo caso l'assegnamento X^* sarà semplicemente una combinazione tale per cui tutti i carichi possano essere eseguiti senza superare la soglia di P_{max} e rispettando le priorità. Il risultato potrà essere ottenuto ad esempio tramite l'algoritmo di Asynchronous Backtracking. Non sarà dunque un assegnamento ottimo, ma semplicemente il primo ammissibile.

Una seconda strada è quella di modellare il problema top anch'esso come un DCOP, con la stessa formalizzazione discussa sopra ma con un assegnamento X^* diverso: invece che pensare solo a rispettare i vincoli (ad esempio di energia massima), si cerca di minimizzare anche il costo dell'energia, tramite una funzione di costo C in questo modo: $X^* = argmin\ C(x)$.

L'algoritmo non si fermerà dunque al primo assegnamento valido, ma cercherà l'assegnamento che minimizzi il costo.

Anche qua potremmo pensare ad un parametro che farà propendere più verso completare tutti i task, a costo di un maggior consumo energetico, oppure più verso il risparmio energetico, sacrificando alcuni task non primari, in base al valore assunto dal parametro.

Un'altra possibilità è quella di assegnare X* in modo tale da guadagnare il più possibile, perché ad esempio il condominio è dotato di pannelli solari e più energia verrà venduta e maggiore sarà il guadagno.

In altre parole, al posto che minimizzare costi o energia, si massimizza il profitto. Si avrà quindi una funzione G(x), che rappresenta il profitto ottenuto con un determinato assegnamento delle variabili, che verrà usata nel seguente modo: $X^* = argmax\ G(x)$.

Nel caso in cui, invece, il problema bottom restituisca un solo valore, l'intera formalizzazione resterà la medesima di cui sopra (con anche la divisione tra DSCP e DCOP), ma in questo caso verranno popolati i domini delle variabili del livello top nel modo seguente: verrà interrogato N volte (con N arbitrario, in base alla quantità di scelte che vogliamo) ogni singolo problema bottom, passando ogni volta un parametro diverso (di potenza, di tempo, di finestra temporale, combinazioni di questi, eccetera). In questo modo verrà creato un vettore di possibilità per ogni problema bottom, esattamente come se fosse direttamente il problema a restituire più soluzioni. Tale vettore verrà usato nel problema top come già spiegato sopra.

4.3 - Formalizzazioni schematizzate

I diagrammi che seguono cercano di schematizzare il più possibile i vari casi presentati precedentemente nelle formalizzazioni dei problemi. Tutto ciò che segue è dunque la rappresentazione di quanto discusso in precedenza.

4.3.1 – Caso 1, più valori da livello bottom

Caso 1, livelli bottom: ogni livello bottom restituisce un vettore X*, contenente i diversi assegnamenti X*. Di seguito, in Figura 4.1, è rappresentato un singolo livello bottom, ognuno di questi livelli si comporterà in questo modo.

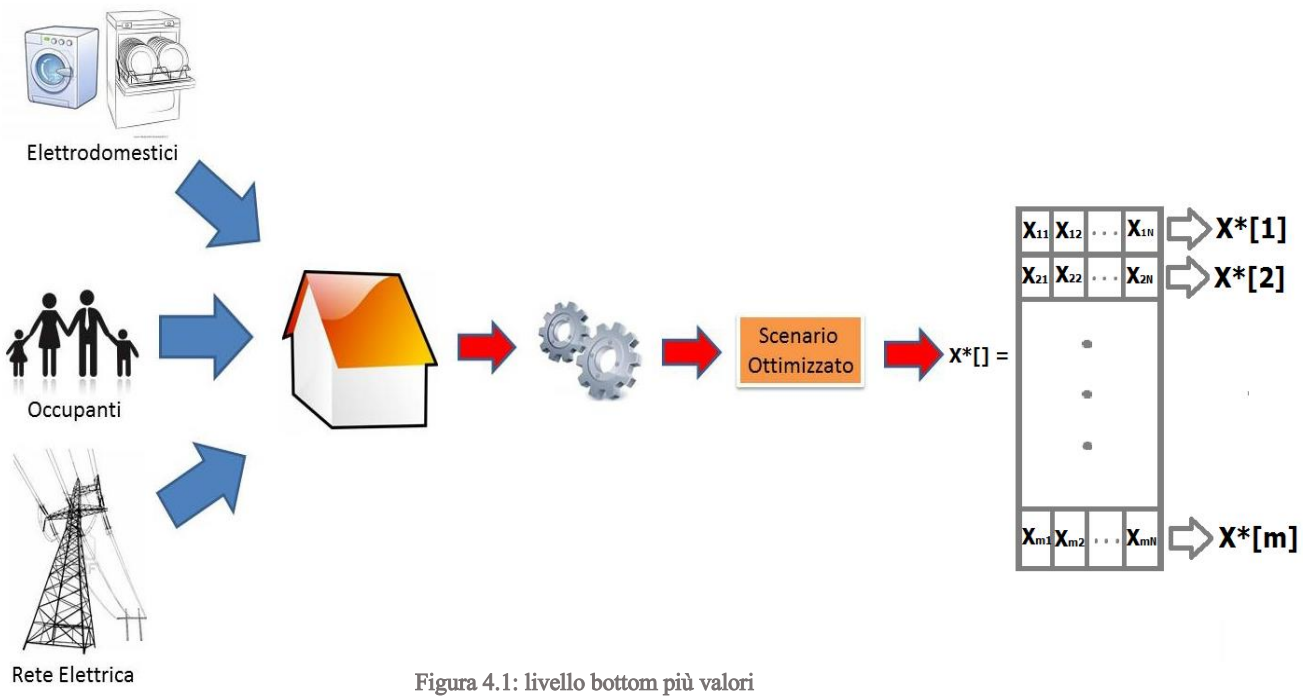


Figura 4.1: livello bottom più valori

In questo caso la lunghezza m del vettore X* dipende dal numero di soluzioni volute, parametro impostato inizialmente (cioè il numero di volte che il livello bottom viene rieseguito). Mentre la lunghezza N di ogni singola soluzione (o cella del vettore X*, che è a sua volta un vettore) dipende dal numero di elettrodomestici.

Caso 1, livello top: Il livello top (Figura 4.2) prende tutti gli X* vettori e compie un'ottimizzazione su essi.

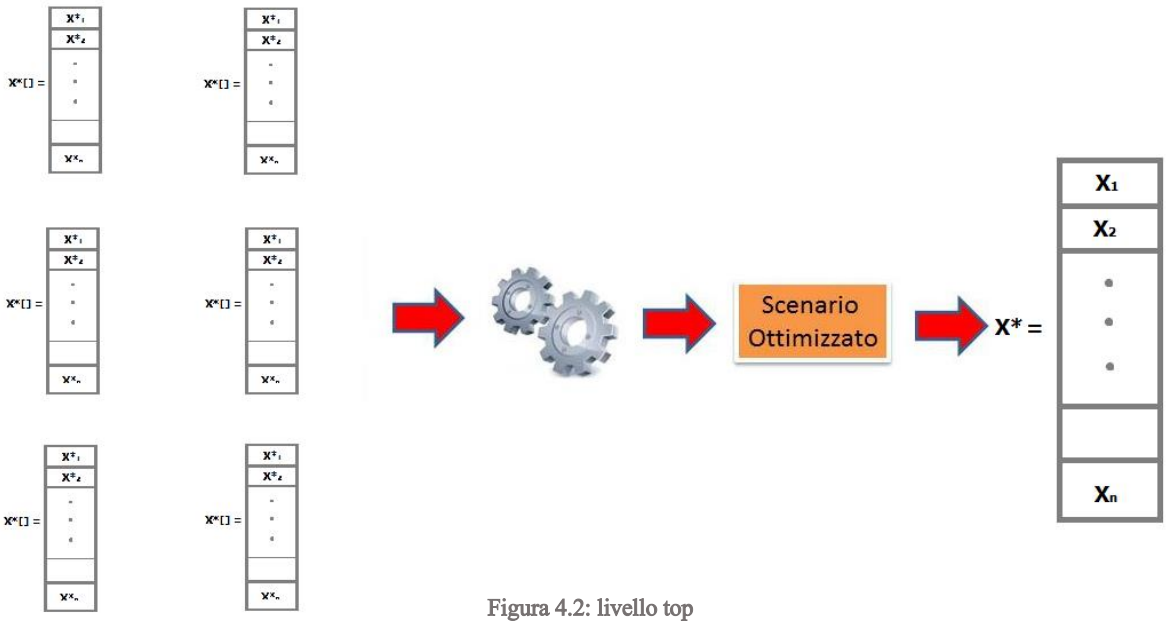


Figura 4.2: livello top

In questo caso la lunghezza n del vettore dipende dal numero di abitazioni presenti nel complesso. Ovviamente nel vettore finale avrò valori che provengono dai diversi vettori X*[] e che ottimizzano lo scenario.

4.3.2 – Caso 2, un solo valore da livello bottom

Caso 2, livelli bottom: ogni livello bottom restituisce un X*, contenente l'assegnamento delle variabili. Di seguito, in Figura 4.3, è rappresentato un singolo livello bottom, ognuno di questi livelli si comporterà in questo modo.

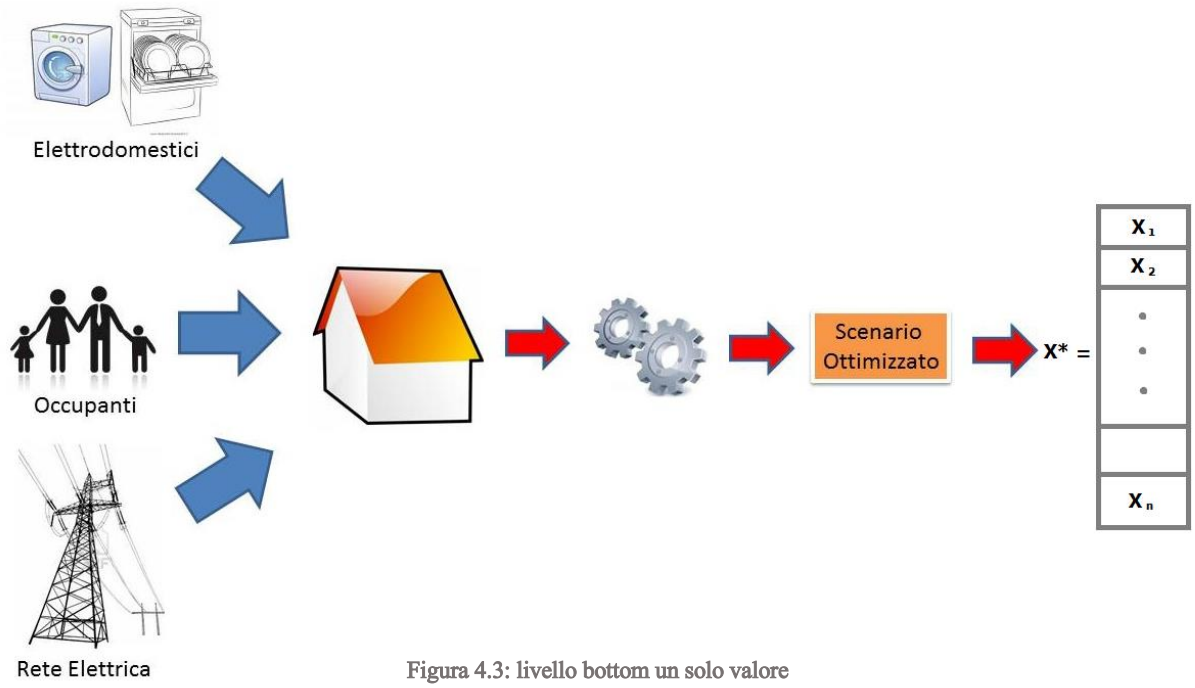


Figura 4.3: livello bottom un solo valore

Tale livello bottom verrà rieseguito più volte facendo variare uno o più parametri in modo tale da avere un vettore X*[] per ogni abitazione, come mostrato in Figura 4.4.
In questo caso la lunghezza m del vettore X* dipende dal numero di soluzioni volute, parametro impostato inizialmente (cioè il numero di volte che il livello bottom viene rieseguito), mentre la lunghezza di ogni singola soluzione (o cella del vettore X*, che è a sua volta un vettore) dipende dal numero di elettrodomestici.

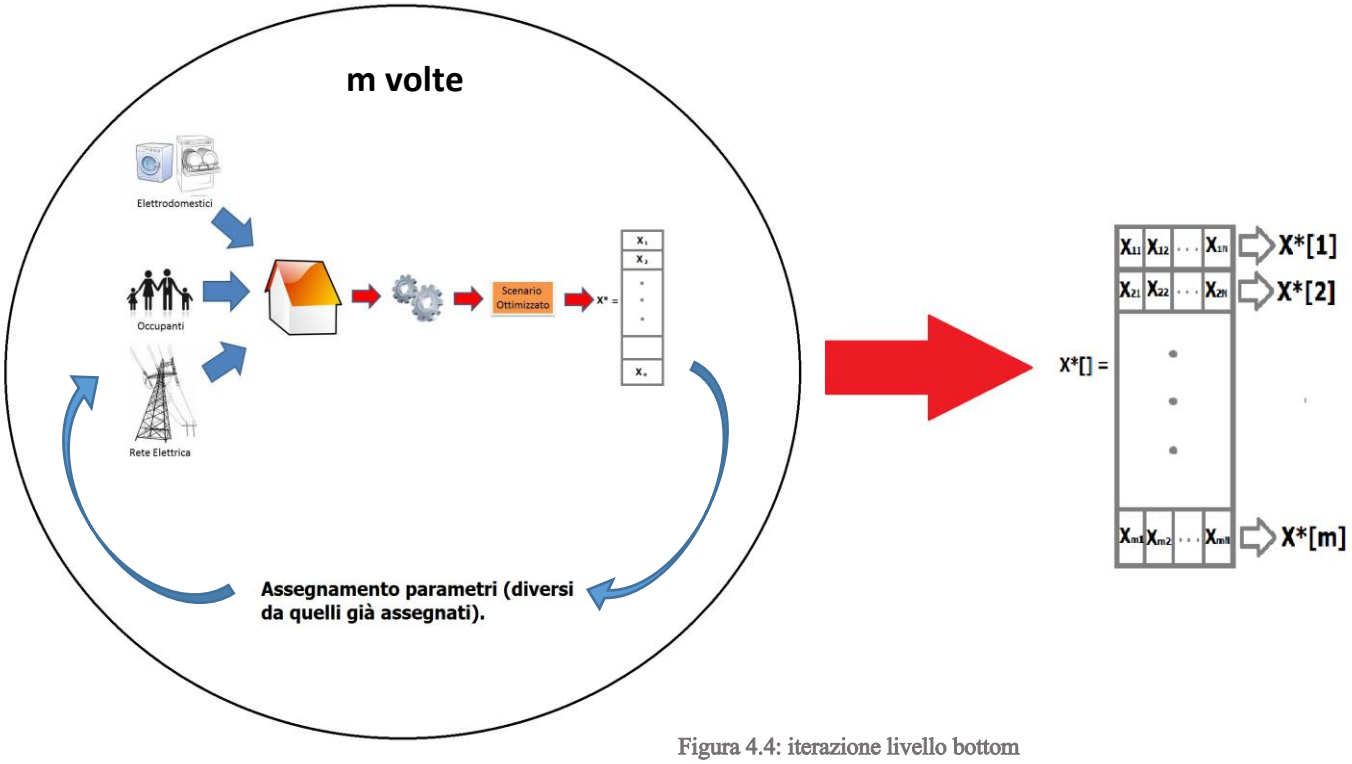


Figura 4.4: iterazione livello bottom

Caso 2, livello top: (Figura 4.5) sarà identico a quello del Caso 1.

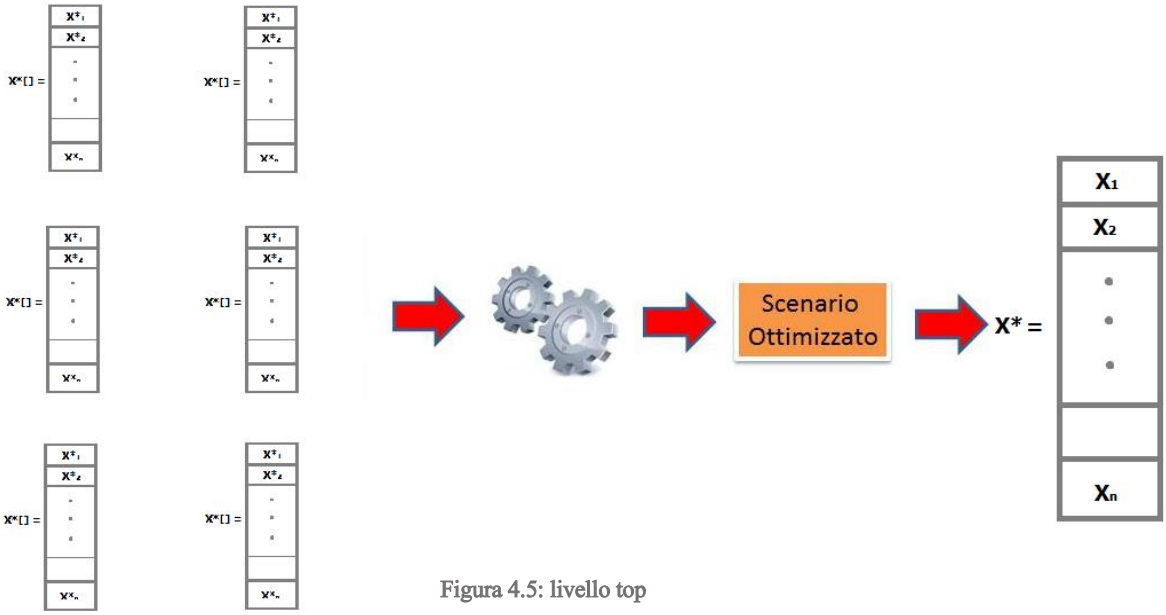


Figura 4.5: livello top

5 – Formulazione del problema applicativo

Nel Capitolo 4 abbiamo introdotto e formalizzato il problema applicativo, spiegando come intendiamo usare una formulazione di tipo DCOP allo scopo di risolvere tale problema, sfruttando il framework FRODO per la sua risoluzione e dunque modellando il DCOP in modo coerente con tale standard.
Di seguito verrà quindi illustrata nel dettaglio l'applicazione sviluppata, come essa interagisce con i sistemi in [10] ed [11], come risolve i problemi a livello bottom e il problema a livello top e il tipo di output prodotto.
Per la parte applicativa è stato utilizzato il linguaggio Java SE 8 (1.8.0) attraverso l'IDE Eclipse, versione Luna 4.4.2.

5.1 I problemi bottom

Per la risoluzione dei problemi bottom, come spiegato in precedenza, si è scelto di utilizzare i sistemi [11] e [10], il primo per la creazione di un ambiente reale in cui un utente richiede di avviare alcuni elettrodomestici in determinati istanti (il numero di elettrodomestici è stato limitato a 2, per evitare l'esplosione dei tempi del programma) mentre il secondo per la creazione di file XCSP validi per FRODO che rappresentano l'appartamento nell'ambiente creato da [11].
Inoltre [10] è stato modificato in modo tale da ricevere in input la potenza massima disponibile per l'appartamento, che sarà il parametro che faremo variare per avere più valori disponibili nei domini degli agenti a livello top.

5.1.1 Formulazione ad attività semplificata

Come spiegato già in [10] la formulazione concettuale dei problemi bottom (e di rimando di quelli top) sarà ad attività semplificata. Con questo tipo di formulazione ad ogni agente (che rappresenta l'elettrodomestico) sono associati tre tipi di variabili: una che indica l'istante d'inizio di un'attività (ogni apparecchio può averne più di una), una che indica la durata di tale attività, e l'altra che ne indica la potenza assorbita. Con il termine attività si intende una fase del profilo di consumo di un carico elettrico che

richieda un assorbimento di potenza (all'incirca) costante. Un esempio è mostrato in Figura 5.1 dove si possono individuare tre attività, ognuna caratterizzata da una certa durata e un certo assorbimento di potenza.

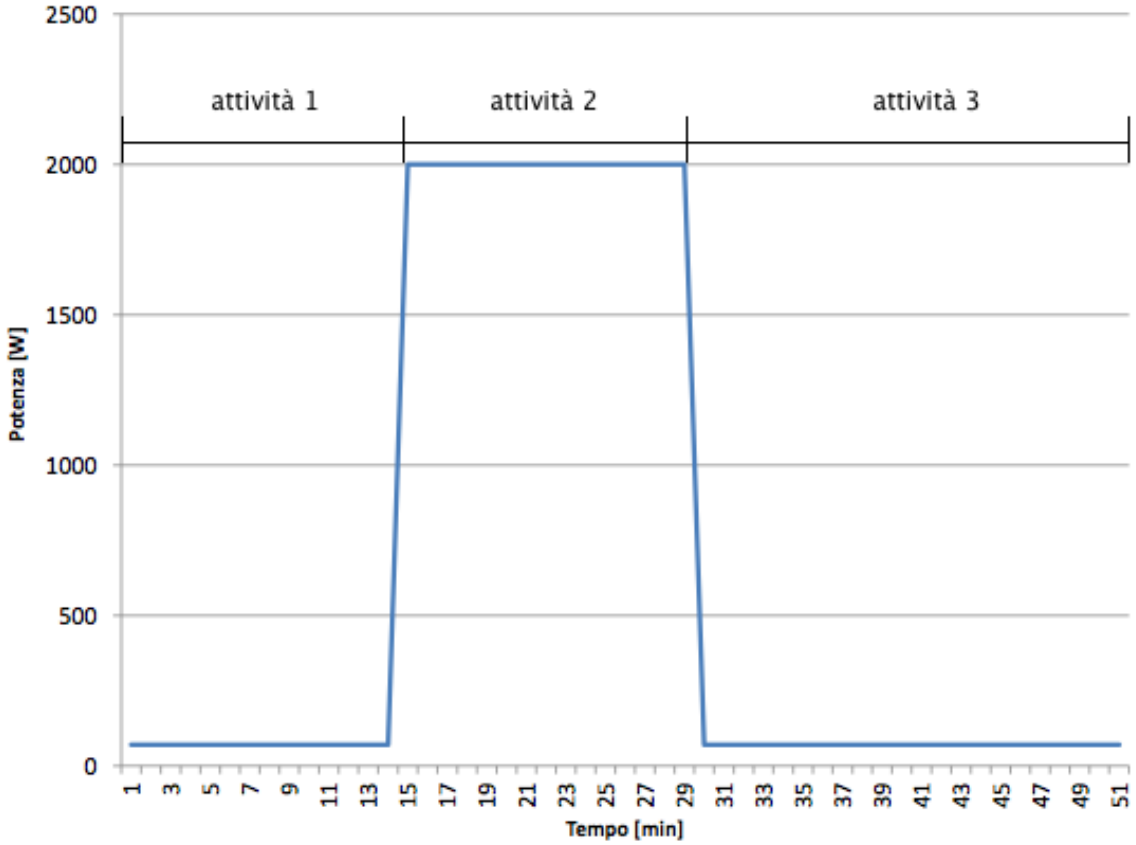


Figura 5.1: esempio di suddivisione in attività di un profilo di consumo

Proseguendo con l'esempio in Figura 5.1, è chiaro come un agente in questo caso controllerebbe 9 variabili in totale, 3 per ogni attività (come detto, istante di inizio dell'attività, durata e potenza di assorbita).

Nel caso delle attività semplificate, le variabili di durata e di potenza assorbita assumeranno un solo valore possibile, perdendo parte della espressività, ma guadagnando in performances e rimanendo comunque applicabile a scenari realistici.

Possiamo quindi formalizzare in modo più preciso il problema come segue:

- $(x_{1,a1}, \dots, x_{1,a_{m1}}, x_{1,d1}, \dots, x_{1,d_{m1}}, x_{1,p1}, \dots, x_{1,p_{m1}}, \dots, x_{n,a1}, \dots, x_{n,a_{mn}}, x_{n,d1}, \dots, x_{n,d_{mn}}, x_{n,p1}, \dots, x_{n,p_{mn}})$: insieme delle variabili controllate dagli *n* agenti, dove m_1 ed m_n indicano dei generici numeri di attività

dei profili di consumo associati. Come spiegato in precedenza, ogni agente controlla tre diverse variabili per attività.

- *($D_{1,1}$, …, $D_{1,m1}$, D_{21}, …, D_{2m2}, D_{n1}, …, D_{nm}):* insieme dei domini delle variabili delle attività sopra descritte. Ogni dominio contiene i possibili istanti di avvio dell'attività. Avremo poi altri 2 domini che rappresenteranno la potenza dissipata e la durata dell'attività.
- Vincoli di continuità tra le attività: rappresentano il collegamento logico tra le variabili di inizio attività. Costringono l'agente ad assegnare come inizio di un'attività l'istante successivo alla fine dell'attività precedente relativa allo stesso carico. Questo vincolo è stato espresso in FRODO tramite un'espressione funzionale complessa inserita all'interno di un predicato (si tratta quindi di un vincolo hard). Segue un esempio:

```
<predicate name="predicate_continuity_agent1">
      <parameters> agent1Act0 agent1Act1 agent1Act2
                   agent1Dur0 agent1Dur1    agent1Dur2 </parameters>
      <expression>
            <functional>and(eq(agent1Act1,add(agent1Act0,agent1Dur0)),
                            eq(agent1Act2,add(agent1Act1,agent1Dur1)))
            </functional>
      </expression>
</predicate>


<constraint name="Constr_continuity_agent1" arity="6" scope="agent1Act0
agent1Act1 agent1Act2 agent1Dur0 agent1Dur1 agent1Dur2 "
reference="predicate_continuity_agent1">
      <parameters>
      agent1Act0 agent1Act1 agent1Act2 agent1Dur0 agent1Dur1 agent1Dur2
      </parameters>
</constraint>
```

La prima parte, come detto in precedenza, è il predicato funzionale che esprime come l'attività 1 debba partire subito aver concluso la 0 ed a sua volta l'attività 2 subito dopo la 1. La seconda parte invece utilizza tale predicato funzionale per imporre il vincolo.

- Vincoli di potenza massima: per ogni quanto di tempo, la somma delle potenze assorbite associate alle attività attive in quel quanto di tempo non deve superare la potenza massima disponibile P_{max}. Questo vincolo è stato espresso in FRODO

attraverso l'utilizzo di vincoli hard enunciati tramite relazioni soft, assegnando un costo infinito alle tuple vietate. Segue un esempio:

```
<relation name="relationagent1Pow0agent0Pow0" arity="4" nbTuples="88" semantics="soft" defaultCost="0"> infinity: 0 30 0 60 </relation>

<constraint name="Constr_agent1Pow0agent0Pow0" arity="4" scope="agent1Act0 agent1Dur0 agent0Act0 agent0Dur0" reference="relationagent1Pow0agent0Pow0"/>
```

In questo caso si sta vietando che agente 1 ed agente 2 partano entrambi all'istante 0, in quanto nell'esempio la potenza massima sarebbe superata.

Si nota come la potenza massima P_{max} sia implicita, in quanto nel vincolo non appare, ma è il programma stesso che crea il file XML a conoscerla e a vietare le combinazioni che supererebbero tale valore.

- Funzioni di costo: per ogni agente, il costo associato f_i, con $i \in [1, n]$, equivale alla somma dei contributi di costo generati da ogni attività. Essi dipendono dalle fasce tariffarie in cui le attività degli agenti vengono eseguite, e si possono calcolare come il prodotto tra il valore campionato della potenza e il costo corrispondente della tariffa, pesato secondo la durata dell'attività.
- Funzione obiettivo: la funzione obiettivo da minimizzare è la somma delle funzioni di costo di ogni agente. Non è necessario indicarlo esplicitamente in FRODO, poiché utilizza automaticamente come funzione obiettivo da ottimizzare la somma di tutte le funzioni di costo indicate.

5.1.2 Utilizzo dei file XCSP

All'avvio il programma creerà dunque un ambiente con l'utilizzo di [11], che verrà poi utilizzato da [10] per la generazione di un file XCSP di un singolo appartamento.

Inoltre [10] prenderà in input un file .txt con all'interno i costi orari dell'energia elettrica ed un file che, in base al tipo di elettrodomestico (generato da [11]) indica il modo in cui i cicli dell'elettrodomestico si dividono in attività, la potenza e la durata di ogni attività.

Come accennato prima, [10] è stato modificato in modo da prendere in input anche un valore di potenza massima (che è quindi ora modificabile ad ogni esecuzione), dunque

mantenendo stabile l'ambiente, faremo variare per N volte (numero di valori voluti per l'accensione di ogni elettrodomestico nel singolo appartamento) la potenza massima disponibile (si partirà da un valore basso e lo si alzerà di volta in volta).

Inoltre, essendo in un ambiente con più appartamenti, in base al numero *m* di appartamenti voluti, creeremo M ambienti diversi con [11], in questo modo avremo *m* appartamenti diversi (con avvio di attività in momenti diversi della giornata), ognuno dei quali avrà N diversi file XCSP a rappresentarlo che conterranno dei vincoli di potenza massima diversi l'uno dall'altro. In Figura 5.2 è mostrato uno schema di quanto spiegato fin qui.

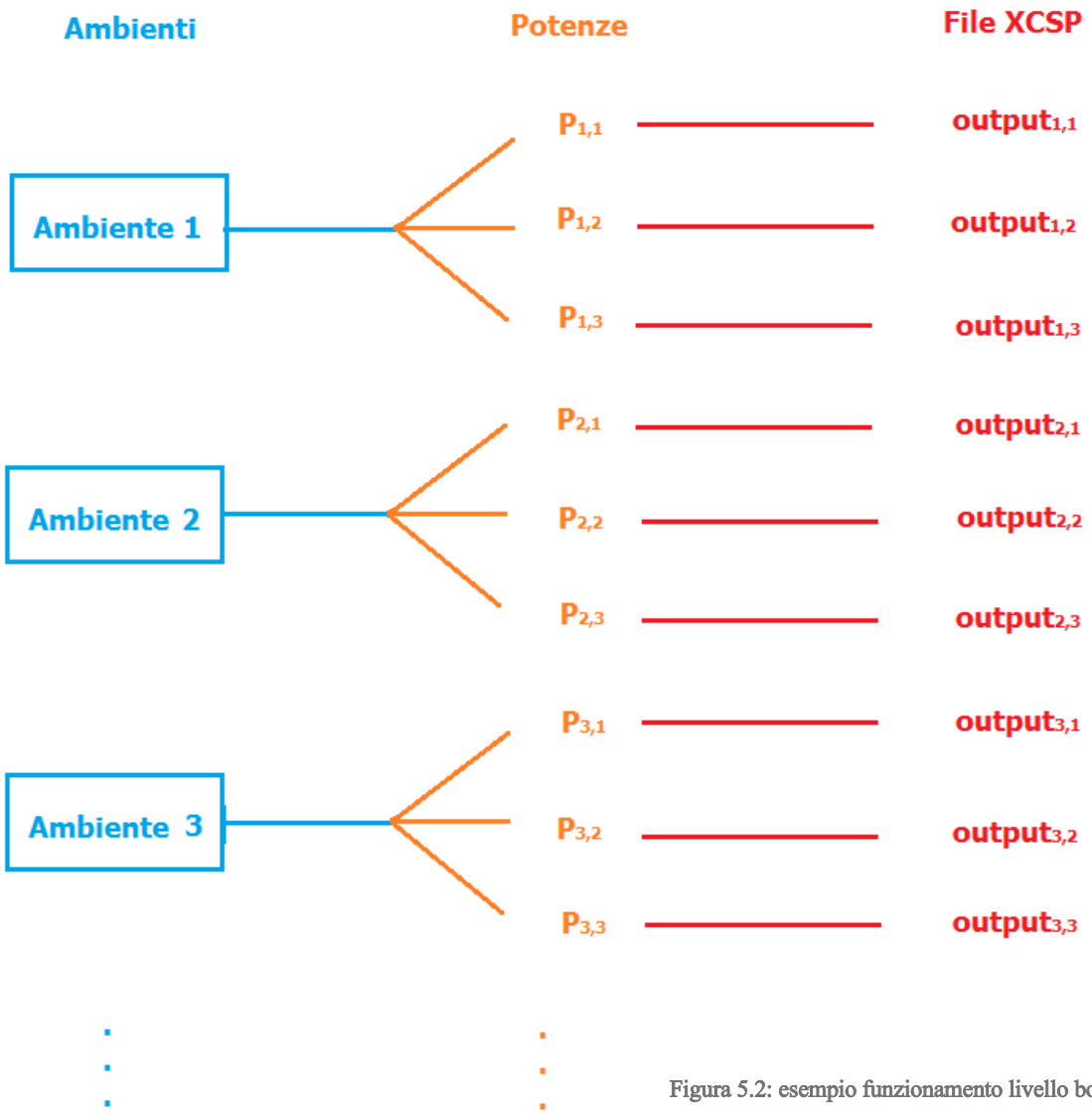


Figura 5.2: esempio funzionamento livello bottom

Gli ambienti sono ovviamente le diverse *m* situazioni reali che servono per creare gli *m* appartamenti, le potenze sono gli N (nell'esempio pari a 3) parametri diversi per ogni

singolo appartamento, in modo da avere più valori possibili nei domini a livello top, e gli output sono i file XCSP, ognuno dei quali rappresenta un singolo appartamento con un determinato valore di potenza massima.
Gli output vengono quindi eseguiti in Java tramite la libreria messa a disposizione da FRODO (libreria frodo2) e vengono risolti attraverso gli algoritmi: DPOP, ADOPT o MGM.
A questo punto il programma ha tutte le variabili necessarie per creare il file XCSP corrispondente al livello top.

5.2 Il problema top

Il problema top ha quindi tutti i valori disponibili per il dominio (i valori di avvio dell'attività, di potenza dissipata e di tempo per cui l'attività resta avviata), il passo seguente è creare il file XCSP che rappresenterà il problema DCOP del livello top, cioè il problema a livello più alto.

5.2.1 Formulazione del problema top

La formulazione del problema top segue quella ad attività semplificate spiegata per il problema bottom nella Sezione 5.1.1. Le differenze sono le seguenti:

- Gli agenti in questo caso non saranno più gli elettrodomestici come a livello bottom ma gli appartamenti stessi.
- I domini e le variabili oltre che riferirsi ad agente e numero di attività, si riferiscono anche al numero di appartamenti, quindi ogni appartamento controllerà: numero di attività per ogni elettrodomestico * numero di elettrodomestici * 3 (i 3 tipi di variabili: quanto un attività può iniziare, la sua durata e la sua potenza dissipata).
  Continuando l'esempio in Sezione 5.1.1, nel caso di due elettrodomestici le cui attività sono divise in 3 parti, avremo un totale di 18 variabili e 18 domini per ogni appartamento.
- Vincoli di continuità tra le attività: definiti come in Sezione 5.1.1. In questo caso il vincolo è stato espresso in FRODO tramite vincoli hard enunciati attraverso

vincoli soft, a cui viene assegnato un costo infinito alle tuple vietate, ossia vietando i casi in cui una attività con indice superiore capiti prima di una con indice inferiore. Ci si riferisce ad attività dello stesso elettrodomestico.

- Vincoli di scelta nello stesso appartamento: a livello top ogni appartamento ha più possibili valori in cui avviare un elettrodomestico. Tali valori vengono da ottimizzazioni a livello bottom. In questo caso però i valori di elettrodomestici diversi dello stesso appartamento non sono slegati tra loro, infatti ogni ottimizzazione a livello bottom restituisce una tupla di valori, una per ogni elettrodomestico nell'appartamento.

  Ad esempio in un appartamento con 2 elettrodomestici e con N = 4 ottimizzazioni otterremo a livello bottom una soluzione di questo tipo:
  {(600,700); (650,720); (670,750); (720, 800)}.
  Quindi, pur avendo domini di questo tipo:

  - Appartamento0Elettrodomestico1 = {600, 650, 670, 720}
  - Appartamento0Elettrodomestico2 = {700, 720, 750, 800}

  In realtà la scelta deve essere vincolata alle coppie sopra descritte (ad esempio se per l'elettrodomestico1 scelgo 670 perché molto conveniente, per l'elettrodomestico2 devo obbligatoriamente scegliere 750).

  Per esprimere questo vincolo si è scelto di usare di nuovo un vincolo hard enunciato attraverso vincolo soft a cui viene assegnato un valore infinito alle coppie non ammissibili (ad esempio la soluzione 600,720 avrà un costo infinito).

5.2.2 Schematizzazione del problema e utilizzo del file XCSP.

Segue in Figura 5.3 una schematizzazione del processo a livello top.

A questo punto il file XCSP creato (come formalizzato nella Sezione 5.2.1) è adatto ad essere eseguito dal framework FRODO, che restituirà dunque una soluzione globale, in cui ogni elettrodomestico di ogni appartamento avrà un suo quanto di tempo di avvio, tale per cui non si supererà la potenza massima consentita e tale quanto rientri tra quelli forniti dai problemi bottom.

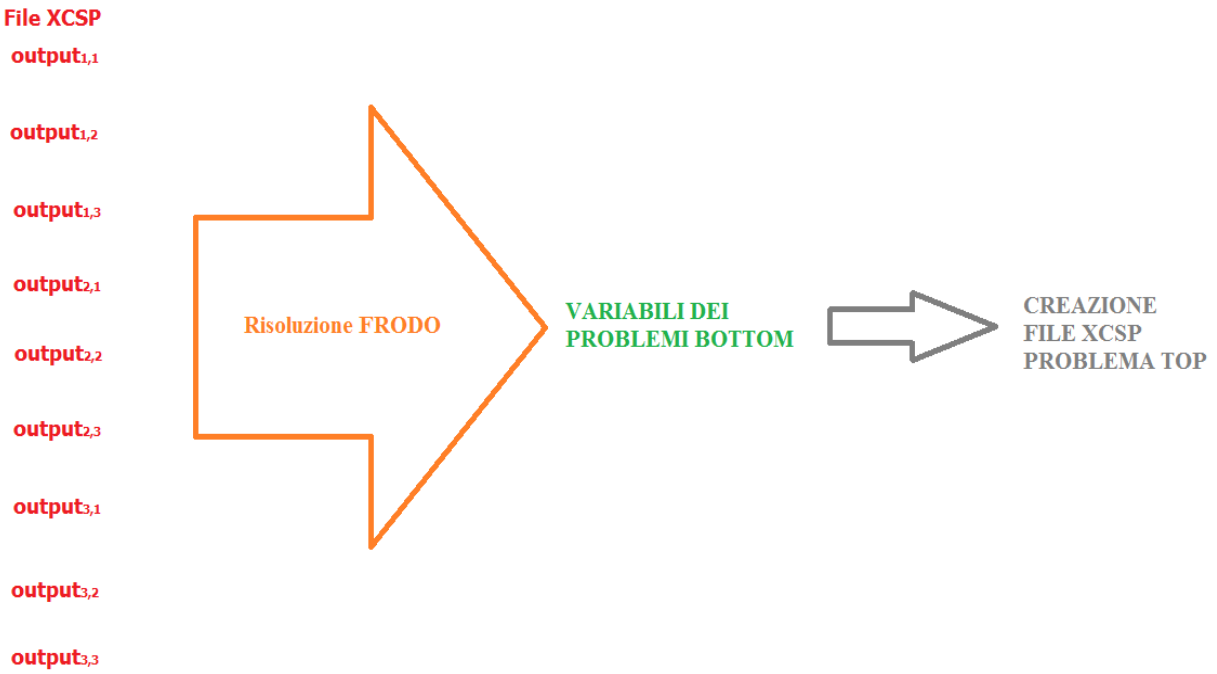


Figura 5.3: schema funzionamento livello top

5.3 Esempio completo di funzionamento

Definiamo, prima di tutto, i file in input per [10]:

- Il file inputTariffe.txt, è un file in cui per ogni ora è indicata la tariffa del costo dell'energia. Il prezzo è espresso in €/kWh. In Figura 5.4 un esempio del file.
- Il file inputPow.txt, che è un semplice file composto da una sola riga con al suo interno il valore della potenza massima.
- Il file inputN che rappresenta il numero di appartamento. Tale file serve per dare il nome all'output di [10], per cui verrà modificato ogni volta che si creerà un nuovo appartamento, in modo crescente da 0 fino ad N (numero di appartamenti).
- Il file inputJ, che rappresenta il numero di problema nello stesso appartamento. Ad esempio se scegliamo di far variare per 4 volte il valore di potenza massima, allora inputJ partirà da 0 e arriverà fino a 3.

START(minuti)	COSTO(€/KWh)
0	0,070
60	0,065
120	0,063
180	0,061
240	0,061
300	0,061
360	0,062
420	0,063
480	0,062
540	0,057
600	0,054
660	0,047
720	0,047
780	0,039
840	0,054
900	0,062
960	0,065
1020	0,075
1080	0,084
1140	0,084
1200	0,075
1260	0,071
1320	0,065
1380	0,063

Figura 5.4: esempio di tariffe in input

- Il file InputScenario.txt, che è il file generato da [11]. Il file è composto 5 colonne:
    - Day: è indicato il giorno della settimana a cui si riferisce l'avvio del ciclo (nel nostro caso il giorno sarà sempre lo stesso, in quanto il nostro lavoro è sviluppato sull'arco della singola giornata).
    - Appliance: indica il tipo di elettrodomestico e il programma di funzionamento, nel nostro caso abbiamo limitato la scelta a lavastoviglie e lavatrice.
    - Start_time: avvio del ciclo, espresso in minuti.
    - Delta_delay: indica l'eventuale presenza di vincoli utente relativi allo spostamento dell'inizio dell'attività.
    - Tmax: indica un vincolo facoltativo dell'utente, che serve per dare un limite entro il quale l'appliance deve terminare.

In Figura 5.5 è espresso un esempio del file.

Day	Appliance	Start_time	Delta_delay	Tmax
0	A2	1130	-2	1440
0	B1	685	-2	1440

Figura 5.5: esempio di scenario in input

- Il file inputConsumi.txt che descrive i profili di consumo associati ai programmi di funzionamento degli elettrodomestici. Per ogni programma di funzionamento sono presenti i seguenti campi:
  - Appliance: che indica la tipologia di elettrodomestico e il programma di funzionamento del ciclo.
  - CycleDuration: che indica la durata del ciclo (espressa in minuti).
  - NbActivities: indica il numero di attività presenti per quel particolare ciclo.
  - ActivityDuration: indica la durata di ogni attività (espressa in minuti).
  - InstantPower: indica la potenza istantanea richiesta da ogni attività, espressa in Watt.

In Figura 5.6 un esempio del file inputConsumi.

```
Appliance       A1
CycleDuration   125
NbActivities    5
ActivityDuration        10 15 85 10 5
InstantPower    62 2120 140 447 88
Appliance       A2
CycleDuration   145
NbActivities    5
ActivityDuration        10 40 70 15 10
InstantPower    64 2047 148 314 678
Appliance       B1
CycleDuration   100
NbActivities    5
ActivityDuration        15 20 40 20 5
InstantPower    70 2000 70 2000 70
Appliance       B2
CycleDuration   135
NbActivities    5
ActivityDuration        15 25 20 60 15
InstantPower    2069 72 2067 71 2073
Appliance       B3
CycleDuration   115
NbActivities    5
ActivityDuration        15 15 60 15 10
InstantPower    70 2000 70 2000 70
```

Figura 5.6: esempio del file inputConsumi.txt

Quindi [10], prenderà in input tali file e creerà un file XCSP, di nome outputN.J. Supponendo che N sia pari a 3 (cioè 3 appartamenti) e J pari a 4 (cioè 4 problemi) avremo degli output di questo tipo (Figura 5.7).

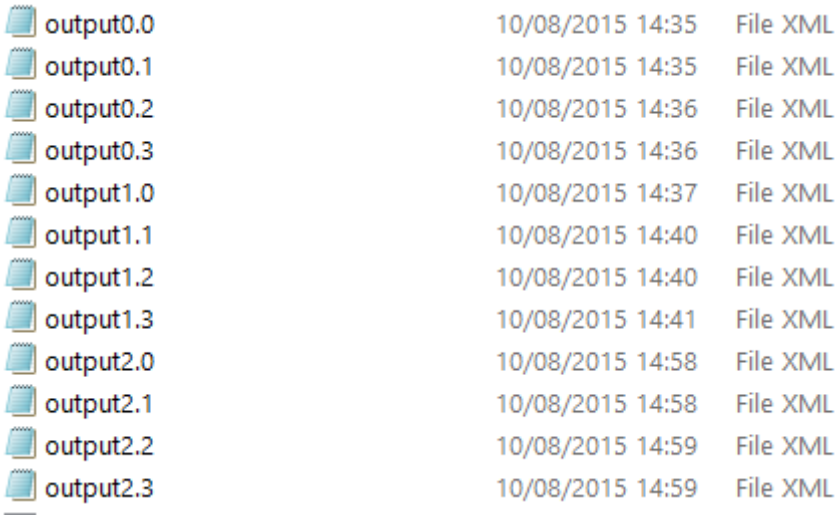


Figura 5.7: esempio di file in output da [10]

Tali file vengono quindi eseguiti, all'interno del programma, attraverso la libreria frodo2, dando origine ai valori del dominio per il livello top. Segue un esempio di risoluzione in Figura 5.8

In prima colonna si ha l'appartamento 1, in seconda l'appartamento 2 ed in terza l'appartamento 3, mentre alla prima riga vi sono le potenze istantanee delle attività, nella seconda l'avvio delle attività ed in terza la durata.

A questo punto il sistema può passare alla creazione del file XCSP di livello top. L'output sarà quindi un file XML formalizzato come in Sezione 5.2.1. In Appendice A.1 è rappresentato il file XCSP che equivale ai domini soprastanti.

Il file XCSP può ora venire usato da FRODO, il quale risolverà il problema DCOP e fornirà un istante di avvio per ogni elettrodomestico di ogni appartamento, ottenendo dunque l'ottimizzazione desiderata, in modo distribuito, facendo interagire il livello top e bottom.

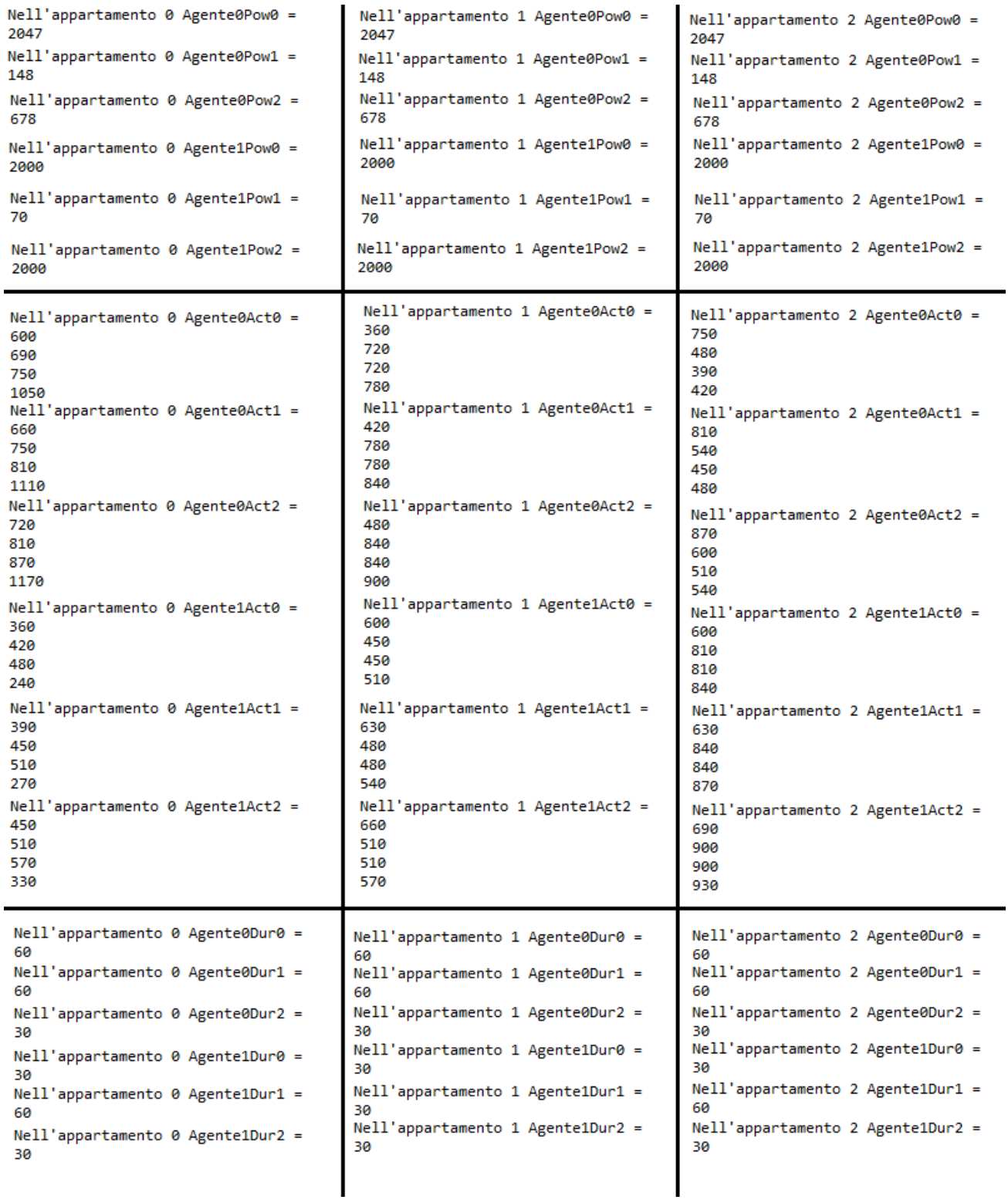


Figura 5.8: risoluzione FRODO dei file XCSP

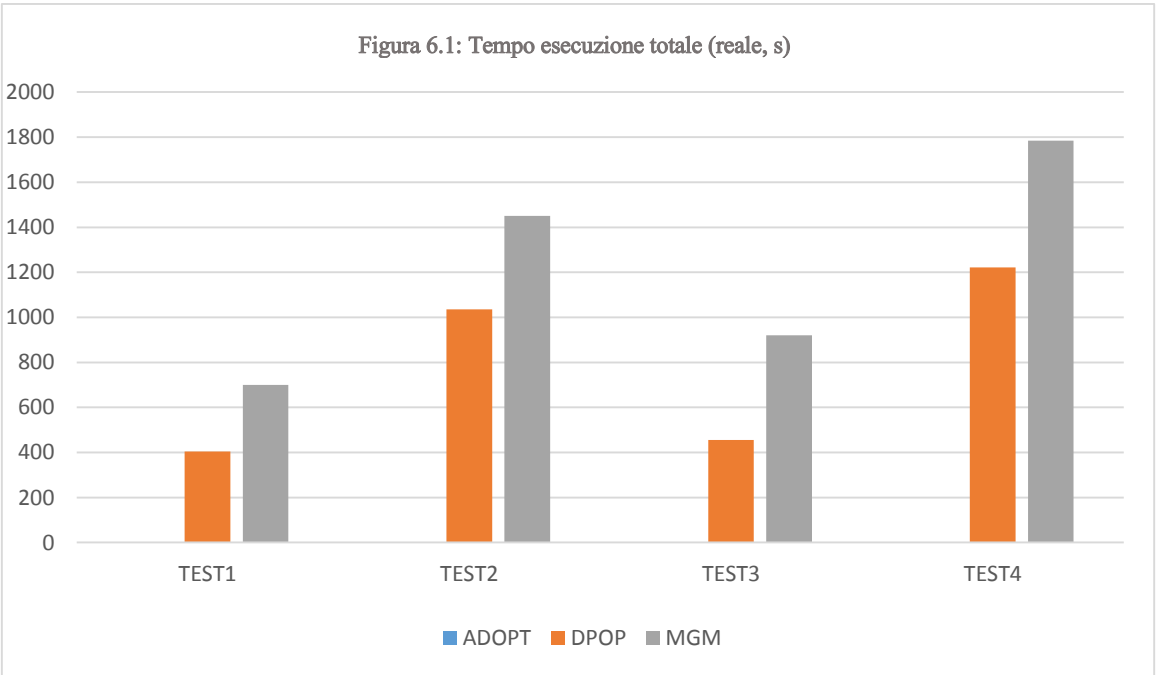
6 – Risultati sperimentali e valutazioni

I valori, ottenuti dai test sperimentali, mostrati in questo capitolo sono stati ottenuti utilizzando il modello descritto in Sezione 5.2 e gli algoritmi di risoluzione descritti in Sezione 3.1, eseguendo FRODO su un singola macchina.

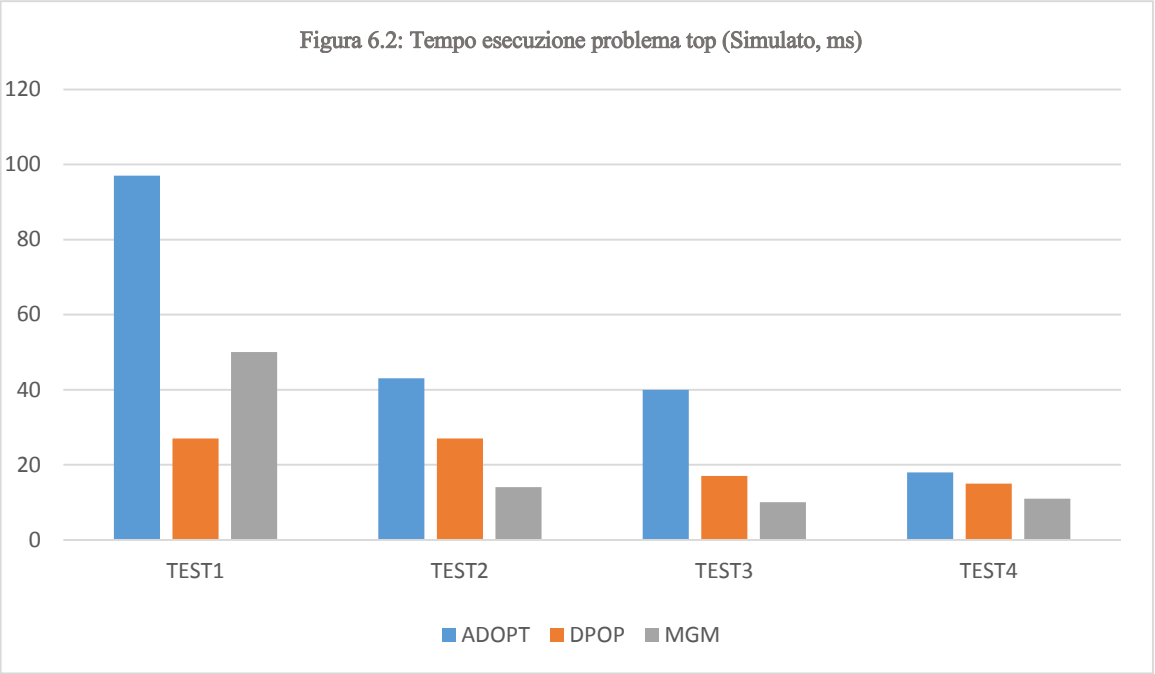
6.1 Ottimizzazione con *n*=2 appartamenti

La prima categoria di test è stata effettuata utilizzando 4 scenari caratterizzati dalla presenza di due appartamenti, i quali sono caratterizzati da ambienti diversi l'uno dall'altro. Il passo di campionamento utilizzato per la definizione dei domini è di 30 minuti. Ogni istanza DCOP associata ad uno scenario è stata risolta tramite gli algoritmi presentati in precedenza.

Figura 6.1: Tempo esecuzione totale (reale, s)



In Figura 6.1 sono rappresentai i test dell'intero sistema in cui si tiene conto del tempo totale (problema bottom più problema top). Si può notare come DPOP sia sempre l'algoritmo più veloce per il calcolo dei problemi. ADOPT non è rappresentato in quanto termina sempre in Time Out (impostato a 6000 secondi).



In Figura 6.2 sono rappresentati i test sull'esecuzione tramite FRODO del solo problema top, formalizzato dal sistema. Ovviamente per ADOPT ed MGM è stato usato il file output.xml creato risolvendo i problemi a livello bottom con DPOP (nel primo caso perché a livello bottom avremmo solo Time Out, nel secondo potenziali soluzioni non ammissibili).

Si può notare come ADOPT in questo caso non vada in Time Out, ma sia comunque sempre l'algoritmo più lento, mentre DPOP ed MGM si avvicinano in quanto a tempistiche, al contrario di quanto riscontrato nei test in Figura 6.1 (probabilmente a causa del minor numero di valori nei domini).

In Figura 6.3 viene invece rappresentato il valore (costo) della soluzione e confrontato con gli altri algoritmi. Il file output.xml è calcolato come nel caso sopra.

Come ci si poteva aspettare il valore per ADOPT e DPOP della soluzione è lo stesso (in quanto entrambi algoritmi completi e ottimi), mentre MGM restituisce un valore della soluzione pari ad infinito (quindi non ammissibile) per la maggioranza dei casi, e solo in un caso riesce a restituire una soluzione ammissibile ma con un costo comunque leggermente superiore agli altri due algoritmi.

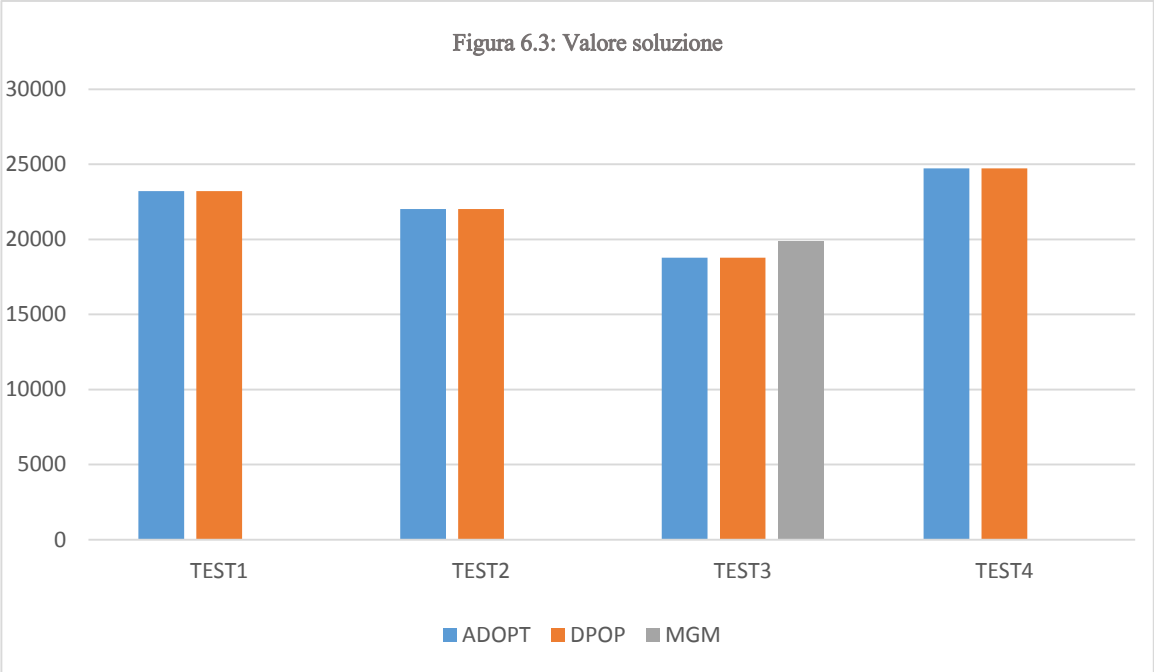


Figura 6.3: Valore soluzione

6.2 Ottimizzazione con *n*=3 appartamenti

La seconda categoria di test è stata effettuata utilizzando 6 scenari caratterizzati dalla presenza di tre appartamenti, i quali sono caratterizzati da ambienti diversi l'uno dall'altro. E' bene precisare che in questo caso per i primi 4 scenari si è scelto di creare dei file xml con dei problemi bottom ad hoc e non casuali in modo da ottenere test mirati e che finissero in tempi brevi (per questo i tempi totali potrebbero sembrare inferiori rispetto al caso con *n*=2 appartamenti, in cui invece i test sono stati effettuati con ambienti casuali), mentre gli altri 2 scenari sono stati generati in modo casuale.

In Figura 6.4 sono rappresentati i test dell'intero sistema in cui si tiene conto del tempo totale (problema bottom più problema top). Nel caso del Test6 i dati sono diminuiti di un fattore 10 per non uscire dalla scala, rendendo non visualizzabili gli altri dati.

Si può notare come, similmente al caso *n*=2 appartamenti, DPOP sia sempre l'algoritmo più veloce per il calcolo dei problemi. ADOPT non è mai rappresentato, tranne in un caso, in quanto termina sempre in Time Out (impostato a 6000 secondi).

Il caso in cui ADOPT è riuscito a calcolare i valori per il livello bottom è stato creato ad hoc con pochi valori nel dominio.

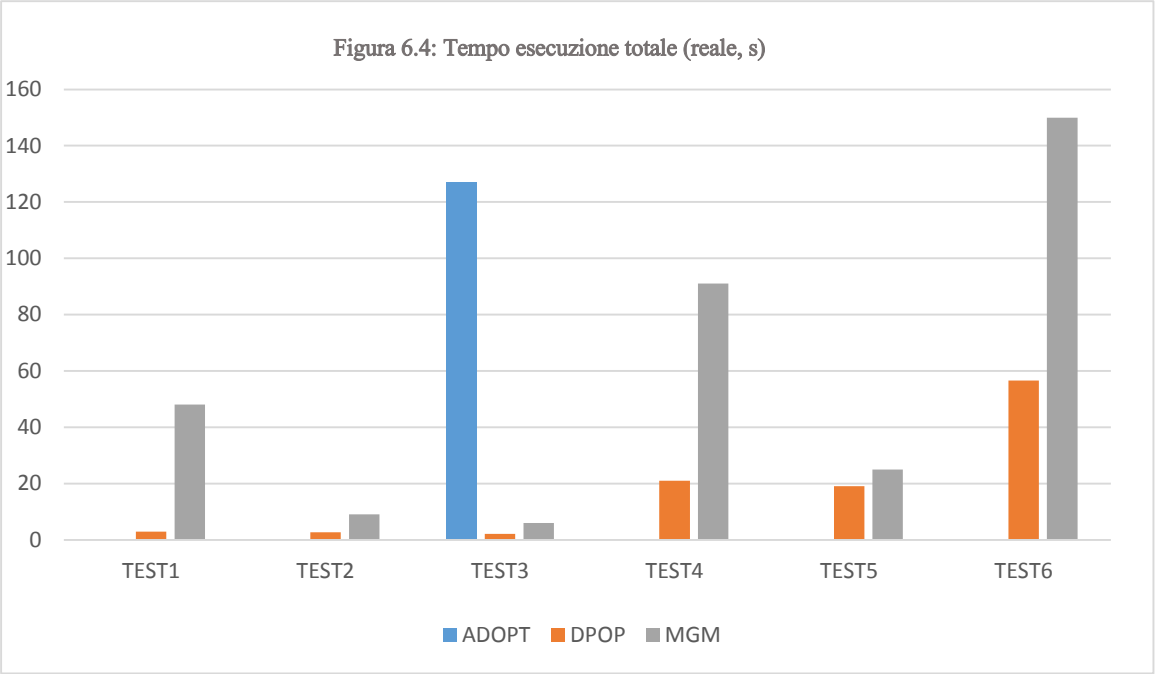


Figura 6.4: Tempo esecuzione totale (reale, s)

In Figura 6.5 sono rappresentati i test sull'esecuzione tramite FRODO del solo problema top, formalizzato dal sistema. Si noti che nel Test5 i valori sono diminuiti di un fattore 10^2 mentre nel Test 6 di un fattore 10 per lo stesso motivo spiegato sopra. Ovviamente per ADOPT ed MGM è stato usato il file output.xml creato risolvendo i problemi a livello bottom con DPOP (nel primo caso perché a livello bottom avremmo solo Time Out, nel secondo potenziali soluzioni non ammissibili) come spiegato in precedenza nei casi n=2.

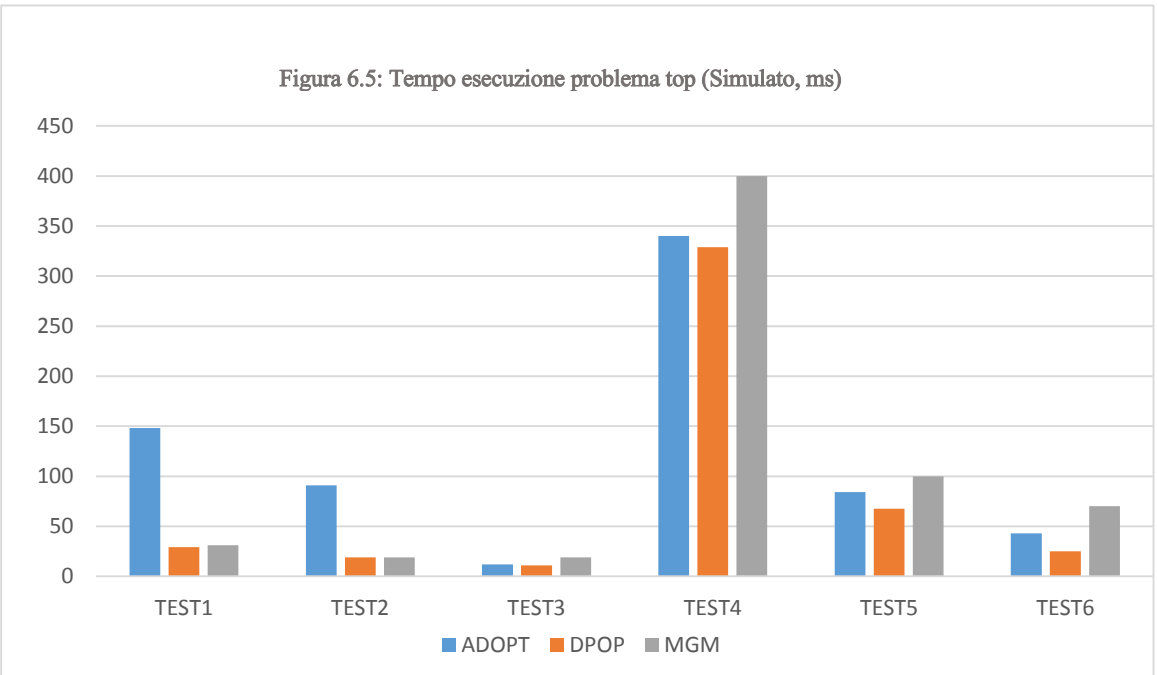


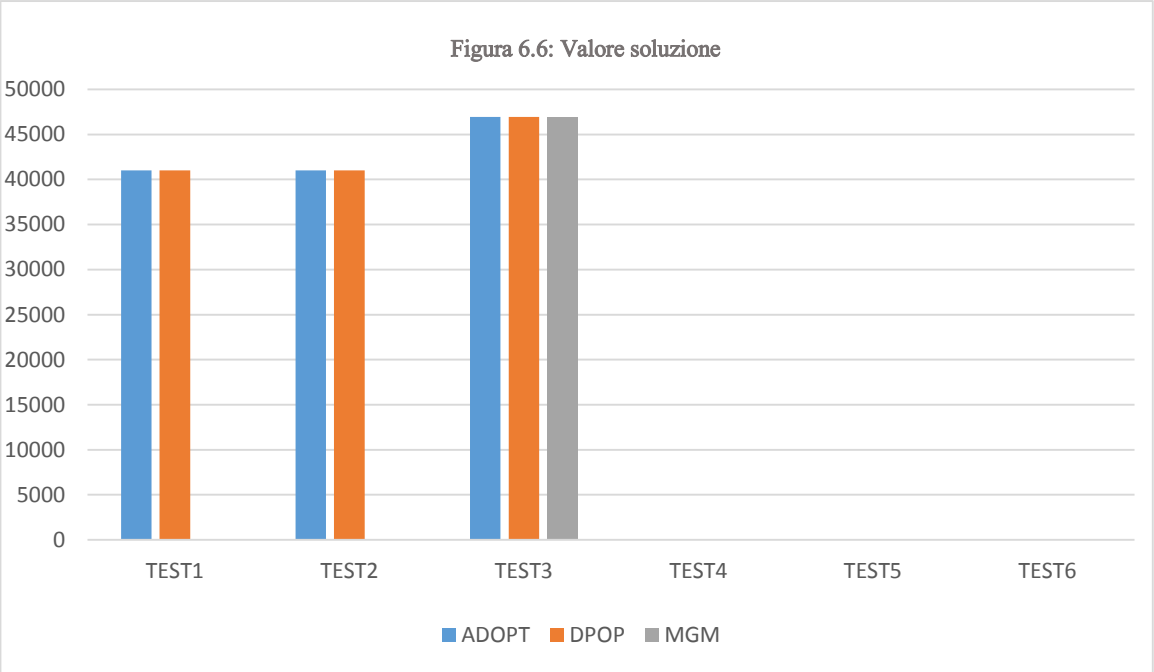
Figura 6.5: Tempo esecuzione problema top (Simulato, ms)

Si può notare come ADOPT in questo caso non vada in Time Out, ma sia sempre l'algoritmo più lento (a parte qualche caso rispetto ad MGM), ma al contrario dei casi con *n*=2 MGM è più lontano da DPOP e più vicino ad ADOPT in termini di tempistiche, cosa causata probabilmente dall'aumento di numero di variabili.

In figura 6.6 viene invece rappresentato il valore della soluzione e confrontato con gli altri algoritmi. Il file output.xml è calcolato come nel caso sopra.

Come ci si poteva aspettare il valore per ADOPT e DPOP della soluzione è lo stesso (in quanto entrambi algoritmi completi e ottimi), mentre MGM restituisce un valore della soluzione pari ad infinito (quindi non ammissibile) per la maggioranza dei casi, e solo in un caso riesce a restituire una soluzione ammissibile che è pari al valore degli altri 2 algoritmi (ma per pura coincidenza).

Figura 6.6: Valore soluzione



Si nota inoltre che gli ultimi 3 test hanno prodotto soluzioni con costo infinito perché qualsiasi valore assegnabile alle variabili superava il valore massimo di potenza istantanea nell'intero edificio (fissata a 3000*numero appartamenti/2).

Anche in questo caso i valori delle soluzioni sono dunque quelli che ci aspettavamo in relazione con i diversi algoritmi.

6.3 Ottimizzazione con *m*=3, 5, 7 livelli di potenza massima per i problemi bottom

La terza categoria di test è stata effettuata utilizzando 5 scenari caratterizzati dalla presenza di tre appartamenti, i quali sono caratterizzati da ambienti diversi l'uno dall'altro. Il passo di campionamento utilizzato per la definizione dei domini è di 30 minuti. Ogni istanza DCOP associata ad uno scenario è stata risolta tramite l'algoritmo DPOP, dimostratosi il migliore nei test precedenti.

In questa categoria di test viene effettuata più volte la stessa ottimizzazione ma facendo variare il numero di livelli di potenza (e quindi il valore di potenza massima data ai problemi bottom, che parte da 2000 e viene aumentata di un fattore 400 ad ogni livello), ottenendo in questo modo un numero sempre maggiore di valori per il dominio delle variabili del problema top.

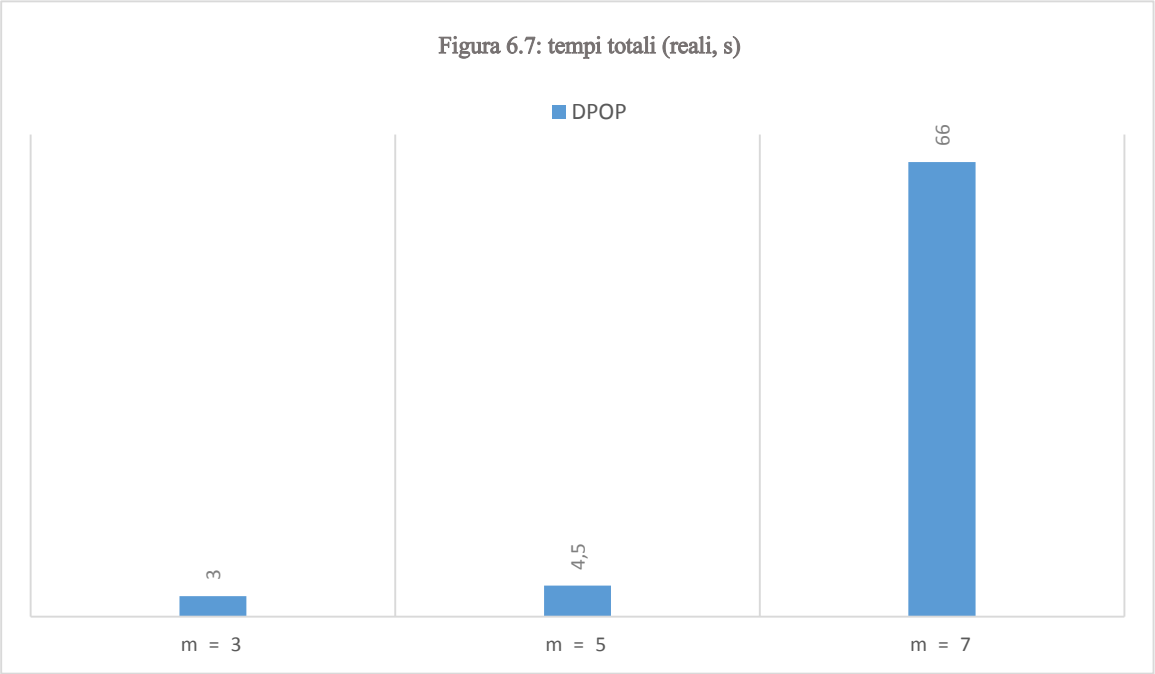
I valori usati sono:

- m = 3, P_{max} = 2000, 2400, 2800
- m = 5, P_{max} = 2000, 2400, 2800, 3200, 3800
- m = 7, P_{max} = 2000, 2400, 2800, 3200, 3800, 4200, 4600

Nei grafici vengono mostrati i valori medi riscontrati da tutti i test.

In Figura 6.7 sono rappresentati i test dell'intero sistema in cui si tiene conto del tempo totale (problema bottom più problema top).

Figura 6.7: tempi totali (reali, s)



Come facilmente prevedibile i test rivelano che all'aumentare del numero di livelli di potenza (e quindi del numero di problemi bottom da risolvere) aumenta anche il tempo di esecuzione.

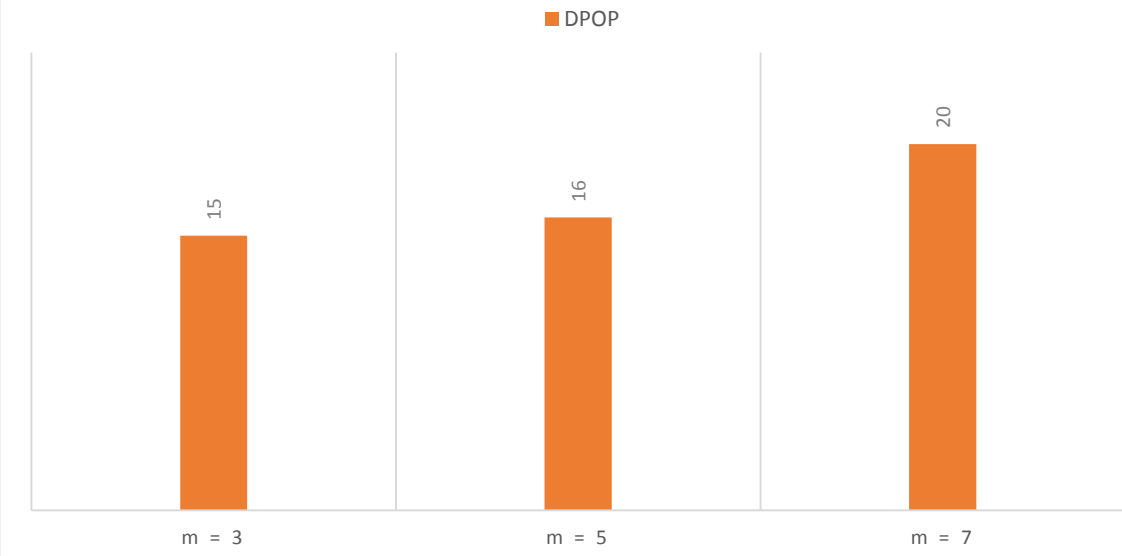
In questo esempio infatti con *m*=3 e 3 appartamenti si hanno in totale 9 problemi bottom, mentre con *m*=7 e 3 appartamenti i problemi bottom diventano 21, con una notevole crescita del tempo di esecuzione.

In Figura 6.8 sono rappresentati i tempi medi per il calcolo del solo problema top.

Da questo test è facile capire come in effetti la causa dell'aumento di tempo totale è da imputarsi principalmente al numero di problemi bottom da risolvere (e dal relativo calcolo di tutti i vincoli per il file xml del problema top) e non alla risoluzione finale del problema top.

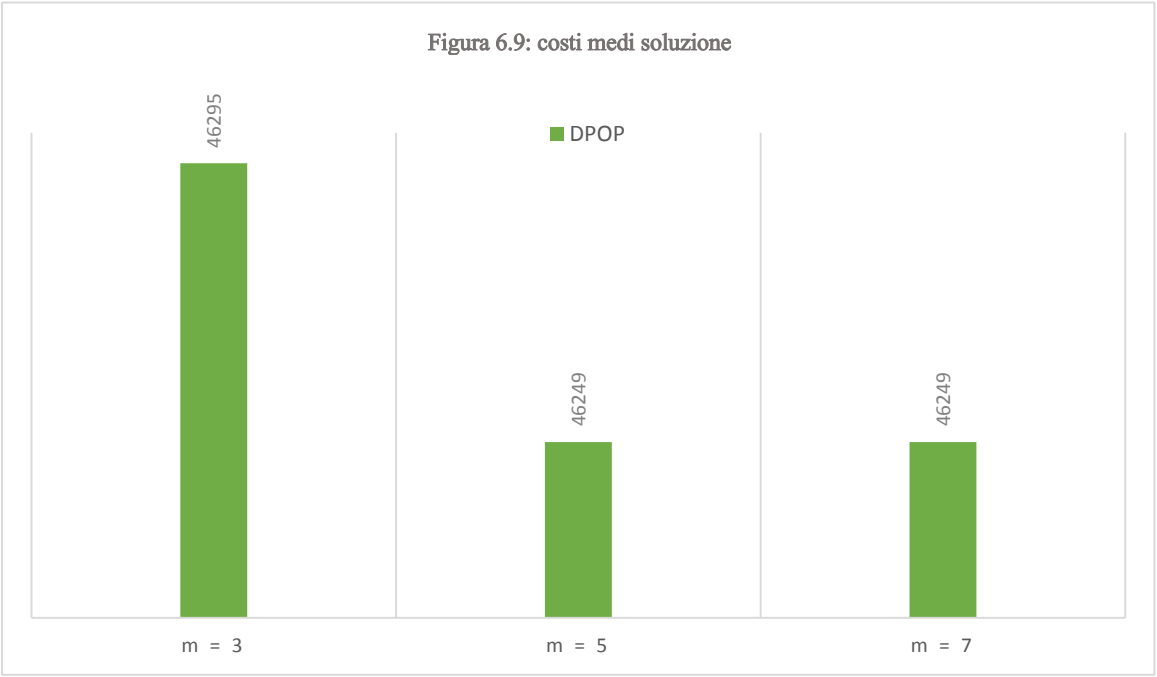
Infatti pur aumentando gradualmente il tempo con l'aumentare del numero di problemi bottom (e quindi dei valori per i diversi domini) tale aumento non è da considerarsi rilevante sui tempi totali di calcolo (il grafico di Figura 6.8 è infatti rappresentato in ms mentre quello di Figura 6.7 in secondi).

Figura 6.8: tempo problema top (simulato, ms)



Infine in Figura 6.9 sono rappresentati i costi medi delle soluzioni.

Figura 6.9: costi medi soluzione



Da tale rappresentazione si può notare un leggero miglioramento del costo della soluzione nel passaggio da *m*=3 ad *m*=5 numero di livelli di potenza. Al contrario il costo della soluzione resta costante nel passaggio da *m*=5 ad *m*=7 livelli.

Da questo si evince che arrivati ad un certo livello di potenza massima (nel caso di *m*=5, 3600 kW/h) non si hanno più vincoli di potenza massima negli ambienti simulati da noi e quindi anche aumentando il valore di potenza massima il risultato non può cambiare.

Ovviamente se la simulazione prevedesse l'avvio di più elettrodomestici il risultato potrebbe cambiare e potremmo ottenere un'ottimizzazione migliore nel caso di *m*=7.

Da questi test si evince comunque che un buon compromesso tra prestazioni e qualità della soluzione si ha con *m* che assume un valore tra 4 e 5.

7 – Conclusioni e sviluppi futuri

Lo scopo di questo lavoro è stato quello di valutare l'uso di una formulazione DCOP per affrontare il problema della gestione dell'energia in piccoli complessi urbani o edifici commerciali con potenze massime limitate.

Il nostro lavoro di tesi differisce da quanto è stato sviluppato fino ad ora perché invece di effettuare un'ottimizzazione ad un singolo livello (e generalmente sul singolo appartamento) essa viene eseguita a 2 livelli:

- Il livello bottom, che rappresenta i singoli appartamenti.
- Il livello top che rappresenta invece l'intero edificio.

I risultati ottenuti dai problemi bottom definiscono il problema top.

Abbiamo quindi sviluppato diversi modelli DCOP per il problema, ne abbiamo scelto uno e abbiamo implementato un sistema che genera un problema così modellato.

In particolare si è scelto di far variare la potenza massima ai livelli bottom e risolvere gli stessi più volte in modo tale da avere diversi valori nel dominio delle variabili di scelta (che rappresentano gli istanti di tempo in cui i carichi dell'appartamento *n* possono avviarsi) a livello top.

Per le valutazioni sperimentali abbiamo utilizzato il framework FRODO a cui viene dato in input un file xml generato dal sistema implementato che rappresenta il problema globale.

Sono stati svolti dei test utilizzando come algoritmi di risoluzione DPOP ed ADOPT (ottimi) ed MGM (non ottimo).

Dai test è stato chiaro come l'algoritmo di risoluzione migliore sia DPOP, con tempi di risoluzione inferiori a quelli degli altri algoritmi e soluzioni migliori (o al più pari a quelle di ADOPT).

Abbiamo mostrato come i tempi di risoluzione aumentano all'aumentare dei livelli *m* di potenza massima ai livelli bottom (e quindi all'aumentare del numero di volte che i problemi bottom vengono rieseguiti).

Con tali test siamo inoltre riusciti a mostrare che nei semplici casi da noi simulati (quindi con un numero massimo di 2 elettrodomestici per appartamento che si avviano una sola volta nell'arco della giornata) la combinazione più vantaggiosa tra tempo di esecuzione e costo della soluzione si ha con *m*, numero di livelli di potenza, pari a 4 o 5.

A seguire vengono presentate alcune possibilità di sviluppi futuri per questo lavoro di tesi.

Un primo sviluppo potrebbe essere quello di utilizzare l'altro modello che abbiamo teorizzato invece di quello poi implementato nella tesi, in cui viene aggiunto un vincolo ad ogni problema bottom in modo da evitare due soluzioni uguali. In questo modo ad ogni iterazione ($1 \dots m$) avrò effettivamente una soluzione diversa dalle precedenti (con il rischio però che tale soluzione non sia ammissibile, in quanto si potrebbe arrivare a vietare tutte le soluzioni ammissibili già trovate in precedenza, ottenendo quindi solo soluzioni non ammissibili procedendo con le esecuzioni).

Un secondo sviluppo sarebbe la modifica di [10] in modo da limitare il dominio degli avvii degli elettrodomestici ad un intorno scelto dall'utente e largo arbitrariamente. In questo modo otterremmo un vantaggio notevole in termini di performances (infatti nel sistema implementato a livello bottom i valori dei domini comprendono tutta la giornata e il sistema sceglie la combinazione migliore, con un inevitabile aumento dei tempi di calcolo).

Con la semplificazione di cui sopra e la relativa diminuzione del tempo di calcolo, inoltre, si potrebbe pensare ad un terzo livello allo scopo di unire l'ottimizzazione di più edifici (ad esempio degli edifici di un intero quartiere).

Bibliografia

[1] **Titolo:** Agent-Based Control for Decentralised Demand Side Management in the Smart Grid.
Autori: Sarvapali D. Ramchurn, Perukrishnen Vytelingum, Alex Rogers, and Nick Jennings.
Sede pubblicazione: Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems – Innovative Applications Track
Anno pubblicazione: 2011
Numero di pagine: 5 - 12

[2] **Titolo:** An Optimal Distributed Constraint Optimisation Algorithm for Efficient Energy Management
Autori: Yoseba K. Penya, Nicholas R. Jennings, Gustaf Neumann
Sede pubblicazione: Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce
Anno pubblicazione: 2005
Numero di pagine: 177 - 182

[3] **Titolo:** TESLA: an extended study of an energy-saving agent that leverages schedule flexibility
Autori: Jun-young Kwak, Pradeep Varakantham, Rajiv Maheswaran, Yu-Han Chang, Milind Tambe, Burcin Becerik-Gerber, WendyWood
Sede pubblicazione: Auton. Agent Multi-Agent Syst.
Anno pubblicazione: 2013
Numero di pagine: 605 - 636

[4] **Titolo:** Distributed Multi-Period Optimal Power Flow for Demand Response in Microgrids
Autori: Paul Scott, Sylvie Thiébaux
Sede pubblicazione: International Workshop On Optimisation In Multi-Agent System (OPTMAS)
Anno pubblicazione: 2015
Numero di pagine: 1 – 15 (Workshop)

[5] **Titolo:** Managing Power Flows in Microgrids Using Multi-Agent Reinforcement Learning
Autori: Fabrice Lauri, Gillian Basso, Jiawei Zhu, Robin Roche, Vincent Hilaire, Abderrafiâa Koukam
Sede pubblicazione: Agent Technologies in Energy Systems (ATES)
Anno pubblicazione: 2013
Numero di pagine: 1 – 8 (Workshop)

[6] **Titolo:** Optimizing the Energy Exchange between the Smart Grid and Building Systems
Autori: Elena Mocanu, Kennedy O. Aduda, Phuong H. Nguyen, Gert Boxem, Wim Zeiler, Madeleine Gibescu, Wil L. Kling
Sede pubblicazione: Proceedings of the 49th international universities power engineering conference (UPEC)
Anno pubblicazione: 2014
Numero di pagine: 1 – 6 (Workshop)

[7] **Titolo:** Building THINC: User Incentivization and Meeting Rescheduling for Energy Savings
Autori: Jun-young Kwak, Debarun Kar, William Haskell, Pradeep Varakantham, Milind Tambe.
Sede pubblicazione: Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems
Anno pubblicazione: 2014
Numero di pagine: 925 – 932

[8] **Titolo:** Demand response for home energy management system
Autori: Yantai Huang, Hongjun Tian, Lei Wang.
Sede pubblicazione: Electrical Power and Energy Systems
Anno pubblicazione: 2015
Numero di pagine: 448 – 455

[9] **Titolo:** State of the World 2013: Is Sustainability Still Possible?
Autori: Worldwatch Institute
Sede pubblicazione: The Worldwatch Institute,
Anno pubblicazione: 2013

[10] **Titolo:** Valutazione sperimentale di formulazioni DCOP per problemi di gestione dell'energia
Autori: Roberto Carettoni
Sede pubblicazione: Tesi di laurea magistrale, politecnico di Milano
Anno pubblicazione: 2013
Numero di pagine: 1 – 94

[11] **Titolo:** Realizzazione e valutazione di un sistema per il demand side management in ambito domestico.
Autori: F. Montecchi
Sede pubblicazione: Tesi di laurea magistrale, politecnico di Milano
Anno pubblicazione: 2012

Appendice A – Esempio di file output.xml

```
<instance>
<presentation name="optimization" maxConstraintArity="0" maximize="false"
format="XCSP
2.1_FRODO"/>
<agents nbAgents="3">
<agent name="app0"/>
<!-- Agente dell'appartamento 0 -->
<agent name="app1"/>
<!-- Agente dell'appartamento 1 -->
<agent name="app2"/>
<!-- Agente dell'appartamento 2 -->
</agents>
<domains nbDomains="66">
<!-- Domini delle potenze istantanee delle attivita -->
<domain name="domain_app0agent0Pow0" nbValues="1">2047</domain>
<domain name="domain_app0agent0Pow1" nbValues="1">148</domain>
<domain name="domain_app0agent0Pow2" nbValues="1">678</domain>
<domain name="domain_app0agent1Pow0" nbValues="1">2000</domain>
<domain name="domain_app0agent1Pow1" nbValues="1">70</domain>
<domain name="domain_app0agent1Pow2" nbValues="1">2000</domain>
<domain name="domain_app0agent1Pow3" nbValues="1">71</domain>
<domain name="domain_app0agent1Pow4" nbValues="1">2073</domain>
<domain name="domain_app1agent0Pow0" nbValues="1">2120</domain>
<domain name="domain_app1agent0Pow1" nbValues="1">140</domain>
<domain name="domain_app1agent0Pow2" nbValues="1">447</domain>
<domain name="domain_app1agent1Pow0" nbValues="1">2000</domain>
<domain name="domain_app1agent1Pow1" nbValues="1">70</domain>
<domain name="domain_app1agent1Pow2" nbValues="1">2000</domain>
<domain name="domain_app2agent0Pow0" nbValues="1">2120</domain>
<domain name="domain_app2agent0Pow1" nbValues="1">140</domain>
<domain name="domain_app2agent0Pow2" nbValues="1">447</domain>
<domain name="domain_app2agent1Pow0" nbValues="1">2000</domain>
<domain name="domain_app2agent1Pow1" nbValues="1">70</domain>
<domain name="domain_app2agent1Pow2" nbValues="1">2000</domain>
<domain name="domain_app2agent1Pow3" nbValues="1">71</domain>
<domain name="domain_app2agent1Pow4" nbValues="1">2073</domain>
<!-- Domini delle durate delle attivita -->
<domain name="domain_app0agent0Dur0" nbValues="1">60</domain>
<domain name="domain_app0agent0Dur1" nbValues="1">60</domain>
<domain name="domain_app0agent0Dur2" nbValues="1">30</domain>
<domain name="domain_app0agent1Dur0" nbValues="1">30</domain>
<domain name="domain_app0agent1Dur1" nbValues="1">30</domain>
<domain name="domain_app0agent1Dur2" nbValues="1">30</domain>
<domain name="domain_app0agent1Dur3" nbValues="1">60</domain>
<domain name="domain_app0agent1Dur4" nbValues="1">30</domain>
<domain name="domain_app1agent0Dur0" nbValues="1">30</domain>
<domain name="domain_app1agent0Dur1" nbValues="1">90</domain>
<domain name="domain_app1agent0Dur2" nbValues="1">30</domain>
<domain name="domain_app1agent1Dur0" nbValues="1">30</domain>
<domain name="domain_app1agent1Dur1" nbValues="1">60</domain>
<domain name="domain_app1agent1Dur2" nbValues="1">30</domain>
<domain name="domain_app2agent0Dur0" nbValues="1">30</domain>
<domain name="domain_app2agent0Dur1" nbValues="1">90</domain>
<domain name="domain_app2agent0Dur2" nbValues="1">30</domain>
<domain name="domain_app2agent1Dur0" nbValues="1">30</domain>
<domain name="domain_app2agent1Dur1" nbValues="1">60</domain>
```

```
<domain name="domain_app2agent1Dur2" nbValues="1">30</domain>
<domain name="domain_app2agent1Dur3" nbValues="1">60</domain>
<domain name="domain_app2agent1Dur4" nbValues="1">30</domain>
<!-- Domini delle variabili attivita -->
<domain name="domain_app0agent0Act0" nbValues="2">660 690</domain>
<domain name="domain_app0agent0Act1" nbValues="2">720 750</domain>
<domain name="domain_app0agent0Act2" nbValues="2">780 810</domain>
<domain name="domain_app0agent1Act0" nbValues="2">750 660</domain>
<domain name="domain_app0agent1Act1" nbValues="2">780 690</domain>
<domain name="domain_app0agent1Act2" nbValues="2">810 720</domain>
<domain name="domain_app0agent1Act3" nbValues="1">750</domain>
<domain name="domain_app0agent1Act4" nbValues="1">810</domain>
<domain name="domain_app1agent0Act0" nbValues="1">690</domain>
<domain name="domain_app1agent0Act1" nbValues="1">720</domain>
<domain name="domain_app1agent0Act2" nbValues="1">810</domain>
<domain name="domain_app1agent1Act0" nbValues="1">720</domain>
<domain name="domain_app1agent1Act1" nbValues="1">750</domain>
<domain name="domain_app1agent1Act2" nbValues="1">810</domain>
<domain name="domain_app2agent0Act0" nbValues="1">690</domain>
<domain name="domain_app2agent0Act1" nbValues="1">720</domain>
<domain name="domain_app2agent0Act2" nbValues="1">810</domain>
<domain name="domain_app2agent1Act0" nbValues="2">720 660</domain>
<domain name="domain_app2agent1Act1" nbValues="2">750 690</domain>
<domain name="domain_app2agent1Act2" nbValues="2">810 720</domain>
<domain name="domain_app2agent1Act3" nbValues="1">750</domain>
<domain name="domain_app2agent1Act4" nbValues="1">810</domain>
</domains>
<variables nbVariables="66">
<!-- Potenze istantanee attivita -->
<variable name="app0agent0Pow0" domain="domain_app0agent0Pow0" agent="app0"/>
<variable name="app0agent0Pow1" domain="domain_app0agent0Pow1" agent="app0"/>
<variable name="app0agent0Pow2" domain="domain_app0agent0Pow2" agent="app0"/>
<variable name="app0agent1Pow0" domain="domain_app0agent1Pow0" agent="app0"/>
<variable name="app0agent1Pow1" domain="domain_app0agent1Pow1" agent="app0"/>
<variable name="app0agent1Pow2" domain="domain_app0agent1Pow2" agent="app0"/>
<variable name="app0agent1Pow3" domain="domain_app0agent1Pow3" agent="app0"/>
<variable name="app0agent1Pow4" domain="domain_app0agent1Pow4" agent="app0"/>
<variable name="app1agent0Pow0" domain="domain_app1agent0Pow0" agent="app1"/>
<variable name="app1agent0Pow1" domain="domain_app1agent0Pow1" agent="app1"/>
<variable name="app1agent0Pow2" domain="domain_app1agent0Pow2" agent="app1"/>
<variable name="app1agent1Pow0" domain="domain_app1agent1Pow0" agent="app1"/>
<variable name="app1agent1Pow1" domain="domain_app1agent1Pow1" agent="app1"/>
<variable name="app1agent1Pow2" domain="domain_app1agent1Pow2" agent="app1"/>
<variable name="app2agent0Pow0" domain="domain_app2agent0Pow0" agent="app2"/>
<variable name="app2agent0Pow1" domain="domain_app2agent0Pow1" agent="app2"/>
<variable name="app2agent0Pow2" domain="domain_app2agent0Pow2" agent="app2"/>
<variable name="app2agent1Pow0" domain="domain_app2agent1Pow0" agent="app2"/>
<variable name="app2agent1Pow1" domain="domain_app2agent1Pow1" agent="app2"/>
<variable name="app2agent1Pow2" domain="domain_app2agent1Pow2" agent="app2"/>
<variable name="app2agent1Pow3" domain="domain_app2agent1Pow3" agent="app2"/>
<variable name="app2agent1Pow4" domain="domain_app2agent1Pow4" agent="app2"/>
<!-- Durate attivita -->
<variable name="app0agent0Dur0" domain="domain_app0agent0Dur0" agent="app0"/>
<variable name="app0agent0Dur1" domain="domain_app0agent0Dur1" agent="app0"/>
<variable name="app0agent0Dur2" domain="domain_app0agent0Dur2" agent="app0"/>
<variable name="app0agent1Dur0" domain="domain_app0agent1Dur0" agent="app0"/>
<variable name="app0agent1Dur1" domain="domain_app0agent1Dur1" agent="app0"/>
<variable name="app0agent1Dur2" domain="domain_app0agent1Dur2" agent="app0"/>
<variable name="app0agent1Dur3" domain="domain_app0agent1Dur3" agent="app0"/>
<variable name="app0agent1Dur4" domain="domain_app0agent1Dur4" agent="app0"/>
<variable name="app1agent0Dur0" domain="domain_app1agent0Dur0" agent="app1"/>
```

```
<variable name="app1agent0Dur1" domain="domain_app1agent0Dur1" agent="app1"/>
<variable name="app1agent0Dur2" domain="domain_app1agent0Dur2" agent="app1"/>
<variable name="app1agent1Dur0" domain="domain_app1agent1Dur0" agent="app1"/>
<variable name="app1agent1Dur1" domain="domain_app1agent1Dur1" agent="app1"/>
<variable name="app1agent1Dur2" domain="domain_app1agent1Dur2" agent="app1"/>
<variable name="app2agent0Dur0" domain="domain_app2agent0Dur0" agent="app2"/>
<variable name="app2agent0Dur1" domain="domain_app2agent0Dur1" agent="app2"/>
<variable name="app2agent0Dur2" domain="domain_app2agent0Dur2" agent="app2"/>
<variable name="app2agent1Dur0" domain="domain_app2agent1Dur0" agent="app2"/>
<variable name="app2agent1Dur1" domain="domain_app2agent1Dur1" agent="app2"/>
<variable name="app2agent1Dur2" domain="domain_app2agent1Dur2" agent="app2"/>
<variable name="app2agent1Dur3" domain="domain_app2agent1Dur3" agent="app2"/>
<variable name="app2agent1Dur4" domain="domain_app2agent1Dur4" agent="app2"/>
<!-- Attivita -->
<variable name="app0agent0Act0" domain="domain_app0agent0Act0" agent="app0"/>
<variable name="app0agent0Act1" domain="domain_app0agent0Act1" agent="app0"/>
<variable name="app0agent0Act2" domain="domain_app0agent0Act2" agent="app0"/>
<variable name="app0agent1Act0" domain="domain_app0agent1Act0" agent="app0"/>
<variable name="app0agent1Act1" domain="domain_app0agent1Act1" agent="app0"/>
<variable name="app0agent1Act2" domain="domain_app0agent1Act2" agent="app0"/>
<variable name="app0agent1Act3" domain="domain_app0agent1Act3" agent="app0"/>
<variable name="app0agent1Act4" domain="domain_app0agent1Act4" agent="app0"/>
<variable name="app1agent0Act0" domain="domain_app1agent0Act0" agent="app1"/>
<variable name="app1agent0Act1" domain="domain_app1agent0Act1" agent="app1"/>
<variable name="app1agent0Act2" domain="domain_app1agent0Act2" agent="app1"/>
<variable name="app1agent1Act0" domain="domain_app1agent1Act0" agent="app1"/>
<variable name="app1agent1Act1" domain="domain_app1agent1Act1" agent="app1"/>
<variable name="app1agent1Act2" domain="domain_app1agent1Act2" agent="app1"/>
<variable name="app2agent0Act0" domain="domain_app2agent0Act0" agent="app2"/>
<variable name="app2agent0Act1" domain="domain_app2agent0Act1" agent="app2"/>
<variable name="app2agent0Act2" domain="domain_app2agent0Act2" agent="app2"/>
<variable name="app2agent1Act0" domain="domain_app2agent1Act0" agent="app2"/>
<variable name="app2agent1Act1" domain="domain_app2agent1Act1" agent="app2"/>
<variable name="app2agent1Act2" domain="domain_app2agent1Act2" agent="app2"/>
<variable name="app2agent1Act3" domain="domain_app2agent1Act3" agent="app2"/>
<variable name="app2agent1Act4" domain="domain_app2agent1Act4" agent="app2"/>
</variables>
<relations nbRelations="34">
<!-- Relazioni (funzioni di costo monetario) -->
<relation name="relation_app0agent0Act0" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">5772: 660 60|5772: 690 60</relation>
<relation name="relation_app0agent0Act1" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">417: 720 60|381: 750 60</relation>
<relation name="relation_app0agent0Act2" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">793: 780 30|793: 810 30</relation>
<relation name="relation_app0agent1Act0" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">2820: 750 30|2820: 660 30</relation>
<relation name="relation_app0agent1Act1" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">81: 780 30|98: 690 30</relation>
<relation name="relation_app0agent1Act2" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">2340: 810 30|2820: 720 30</relation>
<relation name="relation_app0agent1Act3" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">183: 750 60</relation>
<relation name="relation_app0agent1Act4" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2425: 810 30</relation>
<relation name="relation_app1agent0Act0" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2989: 690 30</relation>
<relation name="relation_app1agent0Act1" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">558: 720 90</relation>
<relation name="relation_app1agent0Act2" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">522: 810 30</relation>
```

```
<relation name="relation_app1agent1Act0" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2820: 720 30</relation>
<relation name="relation_app1agent1Act1" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">180: 750 60</relation>
<relation name="relation_app1agent1Act2" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2340: 810 30</relation>
<relation name="relation_app2agent0Act0" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2989: 690 30</relation>
<relation name="relation_app2agent0Act1" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">558: 720 90</relation>
<relation name="relation_app2agent0Act2" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">522: 810 30</relation>
<relation name="relation_app2agent1Act0" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">2820: 720 30|2820: 660 30</relation>
<relation name="relation_app2agent1Act1" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">180: 750 60|197: 690 60</relation>
<relation name="relation_app2agent1Act2" arity="2" nbTuples="2" semantics="soft"
defaultCost="infinity">2340: 810 30|2820: 720 30</relation>
<relation name="relation_app2agent1Act3" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">183: 750 60</relation>
<relation name="relation_app2agent1Act4" arity="2" nbTuples="1" semantics="soft"
defaultCost="infinity">2425: 810 30</relation>
<!-- Relazioni relative ai vincoli di scelta variabile -->
<relation name="relation_app0agent0Act0app0agent1act0" arity="2" nbTuples="2"
semantics="soft" defaultCost="0">infinity: 660 660| 690 750</relation>
<relation name="relation_app0agent0Act1app0agent1act1" arity="2" nbTuples="2"
semantics="soft" defaultCost="0">infinity: 720 690| 750 780</relation>
<relation name="relation_app0agent0Act2app0agent1act2" arity="2" nbTuples="2"
semantics="soft" defaultCost="0">infinity: 780 720| 810 810</relation>
<!--
Relazioni relative ai vincoli di continuita delle attivita
-->
<relation name="relationCont_app0agent0Act0app0agent0act1" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 690 720| 690 720| 690 720</relation>
<relation name="relationCont_app0agent0Act1app0agent0act2" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 750 780| 750 780| 750 780</relation>
<relation name="relationCont_app0agent1Act0app0agent1act1" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 750 690| 750 690| 750 690</relation>
<relation name="relationCont_app0agent1Act1app0agent1act2" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 780 720| 780 720| 780 720</relation>
<relation name="relationCont_app0agent1Act2app0agent1act3" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 810 750| 810 750| 810 750</relation>
<relation name="relationCont_app2agent1Act0app2agent1act1" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 720 690| 720 690| 720 690</relation>
<relation name="relationCont_app2agent1Act1app2agent1act2" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 750 720| 750 720| 750 720</relation>
<relation name="relationCont_app2agent1Act2app2agent1act3" arity="2" nbTuples="3"
semantics="soft" defaultCost="0">infinity: 810 750| 810 750| 810 750</relation>
<!--
Relazioni relative ai vincoli di potenza massima per istante
-->
<relation name="relation_maxPow0" arity="3" nbTuples="1" semantics="soft"
defaultCost="0">infinity: 660 690 690</relation>

<constraints nbConstraints="34">
<!--
Vincoli soft relativi alle funzioni di costo monetarie
-->
<constraint name="Constr_app0agent0Act0" arity="2" scope="app0agent0Act0
app0agent0Dur0"
```

```
reference="relation_app0agent0Act0"/>
<constraint name="Constr_app0agent0Act1" arity="2" scope="app0agent0Act1
app0agent0Dur1" reference="relation_app0agent0Act1"/>
<constraint name="Constr_app0agent0Act2" arity="2" scope="app0agent0Act2
app0agent0Dur2"
reference="relation_app0agent0Act2"/>
<constraint name="Constr_app0agent1Act0" arity="2" scope="app0agent1Act0
app0agent1Dur0"
reference="relation_app0agent1Act0"/>
<constraint name="Constr_app0agent1Act1" arity="2" scope="app0agent1Act1
app0agent1Dur1"
reference="relation_app0agent1Act1"/>
<constraint name="Constr_app0agent1Act2" arity="2" scope="app0agent1Act2
app0agent1Dur2"
reference="relation_app0agent1Act2"/>
<constraint name="Constr_app0agent1Act3" arity="2" scope="app0agent1Act3
app0agent1Dur3"
reference="relation_app0agent1Act3"/>
<constraint name="Constr_app0agent1Act4" arity="2" scope="app0agent1Act4
app0agent1Dur4"
reference="relation_app0agent1Act4"/>
<constraint name="Constr_app1agent0Act0" arity="2" scope="app1agent0Act0
app1agent0Dur0"
reference="relation_app1agent0Act0"/>
<constraint name="Constr_app1agent0Act1" arity="2" scope="app1agent0Act1
app1agent0Dur1"
reference="relation_app1agent0Act1"/>
<constraint name="Constr_app1agent0Act2" arity="2" scope="app1agent0Act2
app1agent0Dur2"
reference="relation_app1agent0Act2"/>
<constraint name="Constr_app1agent1Act0" arity="2" scope="app1agent1Act0
app1agent1Dur0"
reference="relation_app1agent1Act0"/>
<constraint name="Constr_app1agent1Act1" arity="2" scope="app1agent1Act1
app1agent1Dur1"
reference="relation_app1agent1Act1"/>
<constraint name="Constr_app1agent1Act2" arity="2" scope="app1agent1Act2
app1agent1Dur2"
reference="relation_app1agent1Act2"/>
<constraint name="Constr_app2agent0Act0" arity="2" scope="app2agent0Act0
app2agent0Dur0"
reference="relation_app2agent0Act0"/>
<constraint name="Constr_app2agent0Act1" arity="2" scope="app2agent0Act1
app2agent0Dur1"
reference="relation_app2agent0Act1"/>
<constraint name="Constr_app2agent0Act2" arity="2" scope="app2agent0Act2
app2agent0Dur2"
reference="relation_app2agent0Act2"/>
<constraint name="Constr_app2agent1Act0" arity="2" scope="app2agent1Act0
app2agent1Dur0"
reference="relation_app2agent1Act0"/>
<constraint name="Constr_app2agent1Act1" arity="2" scope="app2agent1Act1
app2agent1Dur1"
reference="relation_app2agent1Act1"/>
<constraint name="Constr_app2agent1Act2" arity="2" scope="app2agent1Act2
app2agent1Dur2"
reference="relation_app2agent1Act2"/>
<constraint name="Constr_app2agent1Act3" arity="2" scope="app2agent1Act3
app2agent1Dur3"
reference="relation_app2agent1Act3"/>
```

```
<constraint name="Constr_app2agent1Act4" arity="2" scope="app2agent1Act4
app2agent1Dur4"
reference="relation_app2agent1Act4"/>
<!--
Vincoli Hard per vietare la scelta di variabili differenti
-->
<constraint name="Constr_app0agent0Act0app0agent1act0" arity="2"
scope="app0agent0Act0
app0agent1Act0" reference="relation_app0agent0Act0app0agent1act0"/>
<constraint name="Constr_app0agent0Act1app0agent1act1" arity="2"
scope="app0agent0Act1
app0agent1Act1" reference="relation_app0agent0Act1app0agent1act1"/>
<constraint name="Constr_app0agent0Act2app0agent1act2" arity="2"
scope="app0agent0Act2
app0agent1Act2" reference="relation_app0agent0Act2app0agent1act2"/>
<!--
Vincoli Hard per vietare la scelta di attivita senza continuita
-->
<constraint name="ConstrCont_app0agent0Act0app0agent0act1" arity="2"
scope="app0agent0Act0
app0agent0Act1" reference="relationCont_app0agent0Act0app0agent0act1"/>
<constraint name="ConstrCont_app0agent0Act1app0agent0act2" arity="2"
scope="app0agent0Act1
app0agent0Act2" reference="relationCont_app0agent0Act1app0agent0act2"/>
<constraint name="ConstrCont_app0agent1Act0app0agent1act1" arity="2"
scope="app0agent1Act0
app0agent1Act1" reference="relationCont_app0agent1Act0app0agent1act1"/>
<constraint name="ConstrCont_app0agent1Act1app0agent1act2" arity="2"
scope="app0agent1Act1
app0agent1Act2" reference="relationCont_app0agent1Act1app0agent1act2"/>
<constraint name="ConstrCont_app0agent1Act2app0agent1act3" arity="2"
scope="app0agent1Act2
app0agent1Act3" reference="relationCont_app0agent1Act2app0agent1act3"/>
<constraint name="ConstrCont_app2agent1Act0app2agent1act1" arity="2"
scope="app2agent1Act0
app2agent1Act1" reference="relationCont_app2agent1Act0app2agent1act1"/>
<constraint name="ConstrCont_app2agent1Act1app2agent1act2" arity="2"
scope="app2agent1Act1
app2agent1Act2" reference="relationCont_app2agent1Act1app2agent1act2"/>
<constraint name="ConstrCont_app2agent1Act2app2agent1act3" arity="2"
scope="app2agent1Act2 app2agent1Act3"
reference="relationCont_app2agent1Act2app2agent1act3"/>
<!--
Vincoli Hard per vietare la scelta di attivita che superano la potenza massima
istantanea
-->
<constraint name="Constr_maxPow0" arity="3" scope="app0agent0Act0 app1agent0Act0
app2agent0Act0 " reference="relation_maxPow0"/>
</constraints>
</instance>
```