Modelling RTP-based Residential Load Scheduling for Demand Response in Smart Grids*

Shan He[1,2], Ariel Liebman[1], Andrea Rendl[1,2], Mark Wallace[1,2], Campbell Wilson[1]

[1] Faculty of IT, Monash University, Australia
[2] National ICT Australia (NICTA) Victoria
{shan.he,ariel.liebman,andrea.rendl,
mark.wallace,campbell.wilson}@monash.edu

Abstract. High electricity demand peaks and uncertain supply from renewable energy sources have a significant impact on the electricity price and the network capacity. One mechanism proposed to tackle this issue is the use of real-time pricing (RTP) at the end customer level. Here electricity retail prices are set in real-time in response to varying supply-demand conditions in a way that reduces peak demand. This way customers have an incentive to switch their consumption to times with low demand. The RTP-based Residential Load Scheduling problem (RTP-RSP) deals with scheduling the customer consumption such that the overall network consumption is balanced, the electricity price is minimized, and customer satisfaction maximized. In this work, we introduce different formulations of the RTP-RSP. We first introduce a formulation where the electricity price is assumed to be known a priori. This model is embedded in a heuristic approach that iteratively adapts the electricity price, based on the actual consumption that is computed by the models. Second, we present a two-stage stochastic optimisation model, where the electricity prices are stochastic. We evaluate both formulations on data based on real-world figures and present some preliminary results.

1 Introduction

Interconnected networks, commonly known as *grids*, deliver electricity from suppliers, such as power plants, to consumers. Since electricity is not stored within the grid, the electricity supply and demand must *balance* at any time and at any point in the network.

Over the past decades electricity demand has grown steadily, while in recent years in Australia and other countries with extreme hot weather conditions, peak demand has grown faster due to recent affordability of reverse-cycle air conditioners, leading to very high demand peaks during the day and low utilisation levels of networks. This puts considerable pressure on electricity power plants as well as on the networks themselves; higher demand requires increased high-voltage transmission network capacity and low-voltage distribution network capacity. However, reinforcing the grid for peak demand, as well as building new power plants, is a costly way to meet increased demand and can lead to a substantial increase in the electricity price. Furthermore, the supply from renewable energy sources, such as solar panels or wind turbines, is a function of

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

external factors such as weather and are out of the operators' control. As a result, supply increases and decreases at unpredictable times, making it more difficult to balance supply and demand.

To address these issues, so-called *smart grids* have been proposed. Smart grids integrate Information System technologies to deliver electricity in a more efficient way. Smart grids would incorporate *smart meters* and *smart appliances* that jointly enable two-way communication between customers and the network operator. For example smart meters record the consumption of the consumer, send it to the network, and receive the pricing information from the network.

Using smart meters, electricity service providers can encourage customers to reduce their electric consumption at certain times by passing through a real-time electricity that reflects the degree to which the network is loaded due to changing demand. The literature refers to schemes like this as *real time pricing* (RTP) [1], under which a dynamic price is generated for every trading period (or timeslot), e.g. half an hourly, based on the aggregated load on the network. This means that the electricity price at a given time depends on the overall electricity consumption in the grid at that time. Customers thereby can reduce their electricity costs by shifting their consumptions to timeslots with a lower price. This process is referred to as *demand response* or *demand management*.

While a fully user interactive scheduling process is feasible, it is in practice impractical and unnecessary under the smart grid paradigm. One of the core promises of the smart grid, is the delegation and automation of the demand response process to *automated demand response systems*. These systems determine the best times to schedule electric appliances, based on the price and the customer's preferences. Customer preferences reflect the inconvenience of shifting electric consumption from a preferred time slot. Thus, the objective is to minimize the customers' inconvenience and their electricity costs. Note that these two objectives are often in conflict: on the one hand, customers often have similar appliances and preferred consumption times (such as using the electric heating in the morning), but on the other hand, the electricity price would be higher if most customers schedule their appliances at the same time.

In this work, we consider the problem of automated demand response for many households within a smart grid. Furthermore, we present models for scheduling appliances in a single household, where the upcoming electricity price is uncertain. We present an global heuristic model where for each household we generate a model that is then integrated into an iterative 'optimisation loop' where in each iteration, the household's schedule is recomputed in response to an updated price forecast, until the overall system converges to minimal overall electricity prices and customer inconvenience. First, we present a deterministic model, where we assume full knowledge of future electricity prices. Second, we propose a stochastic two-stage formulation, where the electricity prices are uncertain for the second half of the day. Furthermore, we report preliminary experimental results on both models.

This paper is organized as follows. First, we give a detailed problem description in Sec. 2. In Sec. 3, we present our iterative heuristic approach to deal with real-time pricing. We continue by introducing deterministic models for the single-household scheduling problem in in Sec. 4, and their stochastic variant in Sec. 5. Finally, Sec. 7 discusses our empirical evaluation and Sec. 8 concludes our work.

2 The RTP-based Residential Load Scheduling Problem

The RTP-based residential scheduling problem (RTP-RSP) is concerned with scheduling electric appliances in households under real time pricing (RTP) to manage peak demand in the grid. The objective is to minimize the overall electricity cost, and to minimize the user inconvenience for many households. The user inconvenience is derived from customer preferences and quantifies how inconvenient it is to shift the operation of an appliance from the preferred time to a different time. In this work, we assume that the user preferences are fully known for all the appliances in their households.

2.1 Formal problem formulation

In the RTP-RSP, the electric appliances are scheduled over a set of discrete time slots $\mathbb{T} = (1, ..., T)$ where each time slot has the same duration τ. The set of households on the network is denoted as $\mathbb{H} = \{h_1, .., h_H\}$ where H denotes the number of households.

Running an electric appliance is defined as a *job*. Each household $h_i \in \mathbb{H}$ has to schedule a set of J jobs $\mathbb{J}_{h_i} = \{j_{i1}, ..., j_{iJ}\}$, where job $j_{ik} \in \mathbb{J}_{h_i}$ is the kth job of household h_i. Each job j_{ik} of household h_i is a tuple $j_{ik} = (c_{ik}, e_{ik}, l_{ik}, q_{ik}, d_{ik}, f_{ik})$, where $c_{ik} \in \mathbb{R}$ is the electric consumption in kilowatt hour (kwh); $e_{ik} \in \mathbb{T}$ is the earliest start time and $l_{ik} \in \mathbb{T}$ is the latest end time; $q_{ik} \in \mathbb{T}$ is the preferred start time; $d_{ik} \in \mathbb{T}$ is the duration (the number of time slots that job j_k will run over); $f_{ik} \in [0, 1]$ is the *"care factor"*, which represents the degree of inconvenience to customers when job j_{ik} is not scheduled at the preferred start time p_{ik}, where '1' represents the maximal inconvenience.

The real time pricing is represented by a dynamic pricing table P. The table contains K consumption thresholds $\hat{c}_1, .., \hat{c}_K$ that each map to a price for each time slot, $\hat{c}_i \rightarrow \{p_{\hat{c}_i 1}, .., p_{\hat{c}_i T}\}$. Thus, if the electricity consumption $\bar{c}$ goes beyond $\hat{c}_i$ where $\hat{c}_i \leq \bar{c} < \hat{c}_{i+1}$, then the price for time slot t corresponds to $p_{\hat{c}_i t}$. An example of a dynamic pricing table is illustrated in Table 1.

A solution to the RTP-RSP is a schedule for each household, summarized in the set $\mathbb{S} = (S_1, .., S_H)$, where S_i is a valid schedule of electric appliances of the ith household. Each schedule S_i corresponds to a list of start times for each job, $S_i = (s_{i1}, ..., s_{iJ})$, where $s_{ik} \in \mathbb{T}$ is the scheduled start time of job j_k of household h_i.

The objective is to minimize each household's electricity bill and inconvenience. For each household, the electricity bill is calculated as:

$$\forall i \in \{1..H\} : bill_i = \sum_{j=1}^{T} p_{aj} * b_{aj} * tc_j, \tag{1}$$

$$b_{aj} == \bar{c}_a <= tc_j < \bar{c}_{a+1} \tag{2}$$

$$tc_j = \sum_{l=1}^{J} ((s_{il} <= j) \wedge (s_{il} + d_{il} - 1 >= j)) * c'_l, \tag{3}$$

$$c'_l = \frac{c_l * 24}{T}. \tag{4}$$

Consumption Threshold [KWh]		Electricity Prices [cents]									
		p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	p_{i6}	p_{i7}	p_{i8}	p_{i9}	p_{i10}
$\hat{c}_1$	15.1	9.9	10.0	10.2	10.3	10.1	9.1	8.8	9.0	8.9	9.0
$\hat{c}_2$	15.4	10.1	10.3	10.5	10.6	10.4	9.4	9.1	9.2	9.2	9.2
$\hat{c}_3$	15.8	10.4	10.5	10.8	10.9	10.7	9.7	9.4	9.5	9.4	9.5
$\hat{c}_4$	16.3	10.7	10.8	11.1	11.2	11.0	9.9	9.7	9.8	9.7	9.8
$\hat{c}_5$	17.0	11.0	11.1	11.3	11.5	11.3	10.2	9.9	10.1	10.0	10.1
$\hat{c}_6$	17.8	11.3	11.4	11.6	11.8	11.5	10.5	10.2	10.4	10.3	10.4
$\hat{c}_7$	18.9	11.5	11.7	11.9	12.0	11.8	10.8	10.5	10.6	10.6	10.6

Table 1. Sample Dynamic pricing table with 7 consumption thresholds for 10 time slots. As an example, if the overall consumption reaches 16.5KWh at timeslot 1, then threshold $\hat{c}_4$ is used to obtain the price, which is $p_{\hat{c}_4,1} = 10.7$.

where p_{aj} is the price of time slot t_j assuming the total consumption of that time slot is greater than $\hat{c}_a$ (the ath consumption threshold); tc_j is the total consumption of time slot t_{jk}; l is the number of job ; s_{il} is the scheduled start time job j_l of household h_i; c'_l is the electric consumption of job j_l in kilowatt per time slot duration τ (which is less than an hour). More specifically, in our work, $\tau = 10$ minutes and $T = 144$ time slots.

The customer inconvenience is calculated as:

$$\forall i \in \{1..H\} : inconv_i = \sum_{k=1}^{J} |q_{ik} - s_{ik}| * f_{ik}, \tag{5}$$

where q_{ik} is the preferred start time of job j_k of household h_i, s_{ik} is the scheduled start time, and f_{ik} is the care factor. The overall objective is to find schedules for all households such that the cost of each household is minimized:

$$\forall i \in \{1..H\} : cost_h = bill_i + inconv_i \tag{6}$$

Example 1. As an example, consider the problem of scheduling 6 electric appliances for a single household. The jobs' parameters are summarized in Table 2. As an example, job-1 consumes 0.9 KWh and takes 5 time slots (50 minutes) to finish. It has to be scheduled between the first (0:00) and the 120th (19:50) time slot, however the customer prefers to schedule it at the 91st time slot (15:00). The care factor is 0, which means the customer does not mind to schedule the job at a different time than the preferred time. A care factor of 1, like for job-5, means that the customer would like to schedule the job as close as possible to the preferred time.

3 Iteratively Improving Electricity Price and Household Schedules

In this work we consider the interleaved objectives of minimizing the electricity price by avoiding demand peaks and minimizing the customer's inconvenience for scheduling their electric appliances. This stems from the real-time pricing (RTP) scheme, where the

electricity price is calculated in real time for each time slot t. In RTP, the price depends on two different factors: the overall consumption and the electricity supply at t. This introduces two issues. First, the overall consumption depends on the household schedules that we seek to compute. This makes scheduling appliances in a single household difficult, since the schedule influences the actual price. Second, the electricity supply is uncertain, since renewable energy sources depend on external factors such as the weather.

We tackle these two issues in the following way. First, we introduce an *iterative market simulator* to deal with the connection between electricity price and actual consumption. The market simulator is an iterative heuristic optimisation approach and discussed in detail below. Second, we consider randomly varying electricity supply as the stochastic component of our problem, which we will discuss in Section 5. For the remaining part of this section however, we assume that the electricity supply is known.

3.1 Iterative Electricity Market Simulator

The iterative market simulator emulates the electricity market and aims to converge to an (local) optimal price via an iterative process. It employs two main components: household energy schedulers and an electricity price calculator.

A household energy scheduler takes a fixed electricity price and computes an optimal schedule for a single household, considering all its appliances and the user preferences. We will refer to this problem as the Household Load Scheduling Problem.

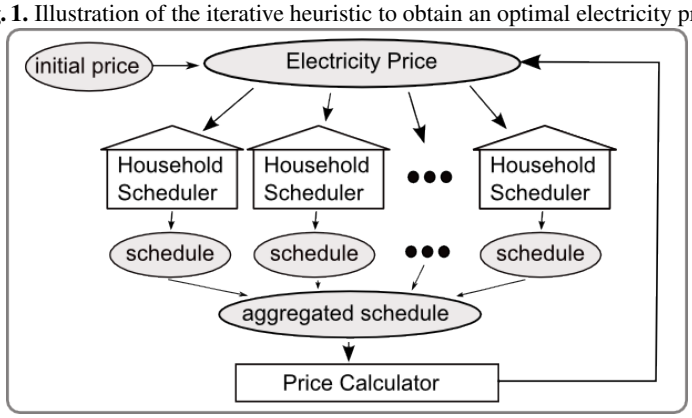
The price calculator takes an aggregated load schedule (the schedules of all households) as input, and uses a dynamic pricing table to calculate the actual electricity price based on those schedules. The dynamic pricing table maps consumption thresholds to electricity prices (for each time slot): if the electricity consumption exceeds the threshold $\hat{c}$ at some time t, then the price can be obtained from $\hat{c}$'s row in table.

The iterative approximation approach of the market simulator is illustrated in Fig. 1. In the first iteration, it intializes the electricity prices to a random value, and then calls all house energy schedulers to schedule their electric appliances against the initial price. This way each house energy scheduler produces a load schedule. In the second step, market simulator produces an aggregated load schedule by summing up the schedules of each house to obtain the total consumptions of each time slot over all houses. Thirdly, the market simulator calls the price calculator to produce a new price schedule for the whole day, based on the aggregated load schedule.

Job parameters	job-1	job-2	job-3	job-4	job-5	job-6
Electric consumption (KWh): c_{ik}	0.9	0.9	2.5	2.5	2.5	2.5
Earliest start time: e_{ik}	1	1	85	85	85	1
Preferred start time: q_{ik}	91	91	97	97	97	97
Duration: d_{ik}	5	5	12	12	12	12
Latest end time: l_{ik}	120	120	132	132	132	132
Care factor: f_{ik}	0	1	0	0.5	1	1

Table 2. Jobs properties of household in Example 1

Fig. 1. Illustration of the iterative heuristic to obtain an optimal electricity price



In the second iteration, the market simulator calls all house energy schedulers to schedule its electric appliances against the new price schedule and continues with this procedure, until an iteration limit has been reached. The idea is that after several iterations, the load schedules of houses will not change, but converge to a local optimal solution for all houses.

4 Deterministic Household Load Scheduling Models

The key components of the market simulator are the household schedulers that compute an optimal schedule for each household, given the current electricity prize. To apply the market simulator on real-world problems (where we schedule appliances of thousands of houses), it is vital that the Household Load Scheduling Problem is solved efficiently, in as little time as possible. Therefore, we now consider this problem and consider its *deterministic* variant.

In the deterministic variant, the household schedulers compute an optimal schedule for a single household, given a fixed electricity price and the customer preferences. This means that we consider the electricity price a function of the demand, and assume that the demand is known. This allows us to facilitate the parameters and preprocess them to slim down the problem models. The parameter preprocessing is described in Sec. 4.1.

4.1 Parameter Preprocessing

During parameter processing we merge certain parameters to faciliate modelling. Since we assume that the electricity price is known a priori, and the customer preferences and job durations are given as a parameter, we can use them to compute the overall *cost* of scheduling an appliance at a specific time slot. More specifically, the cost of scheduling an appliance at time t is defined as: $cost_i = bill_i + inconv_i$.

Furthermore, we *sort* the timeslots by their price in an array that we use in the models to achieve a better search strategy. It allows us to direct the search to try out the cheapest time slots first.

4.2 Constraint Programming Model

We depict the Constraint Programming (CP) model in Fig. 2 in the modelling language MiniZinc [8]. First, we include the global constraints library in line 1, since we are using the global constraint `cumulative`. Then we declare the (preprocessed) problem parameters from line 3- 15: the number of time slots per day (line 3), the number of jobs (line 5), the cost of scheduling a job at a specific timeslot (combining electricity price and the customer inconvenience) at line 11, the time slots ordered by their cheapness in the array `bestTimeSlots`(line 12) where `bestTimeSlots[i,1]=4` states that the cheapest time slot for scheduling job `i` is slot `4`, and `bestTimeSlots[i,2]=7` states that the second-cheapest time slot for job `i` is slot `4`, and so on. Further parameters are the consumption (line 7), duration (line 8), earliest start time (line 9), and latest finish time (line 10) per job, and the overall consumption limit per time slot (line 15).

Then we define the decision variables in lines 18-19. The variables `schedule` represent the timeslot (not ordered chronologically but according to their cheapness) at which each job is scheduled. For instance, `schedule[i] = 3` means that job `i` is scheduled at the 3rd cheapest time slot for that job `i`. This order corresponds to the time slot order obtained during preprocessing and can be easily converted to the actual starting times. The second set of variables, `actualCosts` (line 19), represents the overall cost for each job.

The constraints are listed from line 22-28. First, we link the actual cost to the schedule (line 22). Since the costs are ordered by cheapness and not chronologically, this reflects the actual cost of each job. Second, we impose the `cumulative` constraint on the schedule, stating that the scheduled jobs may not exceed the consumption limit. Finally, we state the search strategy (line 31) and the objective (line 33).

5 A Stochastic Household Load Scheduling Model

In order to model the impact on the market price due to unpredictable and uncontrollable sources of generation (renewable energy sources), we incorporate a stochastic element in our market simulator. Since the majority of these renewable sources are incorporated into wholesale markets as effectively zero variable-cost electricity generators with a stochastic output, they can be treated as subtractive modifiers of the total system demand. This is reflected as a stochastic horizontal shift in the supply curve for every timeslot of the day. The construction of this supply curve is explained is Section 7.

This means that in the *stochastic* variant of the Household Load Scheduling Problem, we consider the *cost* (that incorporates electricity price and the customer incovenience) as uncertain, due to unpredictable fluctuations in supply. We formulate the problem as a two-stage stochastic optimisation problem where we assume that the cost for the first part of the day is known and the cost of the second part is uncertain. This is a first step towards a multi-stage stochastic optimisation problem, where each new time slot corresponds to a stage. We leave the multi-stage formulation for future work.

The model takes N cost scenarios as input, where each scenario represents a possible outcome of the cost of the second half of the day. Furthermore, each scenario is assigned a *weight* that represents its probability of occuring.

Fig. 2. CP model for the household scheduling problem

```
include "globals.mzn";
% ------ input parameters ------------------------------------ %
int: numOfTimeSlots;
set of int: TimeSlots = 1..numOfTimeSlots;
int: numOfJobs;
set of int: Jobs = 1..numOfJobs;
array[Jobs] of int: consumptions;
array[Jobs] of int: durations;
array[Jobs] of int: estarts;
array[Jobs] of int: lends;
array[Jobs, TimeSlots] of int: costs; % price and inconvenience
array[Jobs, TimeSlots] of int: bestTimesSlots; % time slots ordered by cost
int: costLimit = max([costs[j,t]|j in Jobs, t in TimeSlots]);
int: consLimit = min(sum(j in Jobs)(consumptions[j]), timeSlotConsLimit);
float: timeSlotConsLimit;

% --------- variables ----------------------------------------- %
array[Jobs] of var TimeSlots: schedule;
array[Jobs] of var 0..costLimit: actualCosts;

% --------- constraints --------------------------------------- %
constraint forall(j in Jobs)
  (actualCosts[j] = costs[j, schedule[j]]);

constraint forall(j in Jobs)
   (bestTimesSlots[j,schedule[j]] >= estarts[j] /\ bestTimesSlots[j,schedule[j]] +
      durations[j] -1 <= lends[j]);

constraint cumulative([bestTimesSlots[j,schedule[j]]|j in Jobs], [durations[j] -
     1|j in Jobs], consumptions, consLimit);

% -------- objective and search ---------------------------------- %
solve :: seq_search
([ int_search([schedule[j]|j in Jobs], first_fail, indomain_min, complete)])
minimize sum(j in Jobs)(actualCosts[j]);
```

We model the stochastic formulation using Stochastic MiniZinc [9] that can translate the model to its scenario-based deterministic equivalent [10]. We first transformed the CP model from Sec. 4.2 into a stochastic formulation, which we present in Sec. 5.1. Since this formulation had a very poor performance, we re-formulated the model into a MIP-style stochastic model that we present in Sec. 5.2.

5.1 CP stochastic Model

We show excerpts (due to space) of the stochastic CP model in Fig. 3. We start with the parameter declaration (line 2-15), where we first declare the scenarios (line 2) and their weights (line 3). Then we divide the time slots by stage: the ones belonging to stage 1 (line 4) and to stage 2 (line 5). We also divide other parameters by stages, such as the `costs` and `bestTimeSlots`. Similarly, we divide the problem variables by stages (line 17-20). Since we use the deterministic equivalent, the second stage variables are also indexed by scenario.

Fig. 3. Stochastic Two-stage CP formulation of the Household Load Scheduling Problem

```
1  % ------ input parameters ------------------------------------ %
2  set of int: Scenarios = 1..nbScenarios;
3  array[Scenarios] of int: weights;
4  set of int: TimeSlots1 = 1..numOfTimeSlots1;
5  set of int: TimeSlots2 = 1..numOfTimeSlots2;
6  int: costLimit1 = max([costs1[j, t]|j in Jobs, t in TimeSlots1]);
7  int: costLimit2 = max([costs2[sc, j, t]|sc in Scenarios, j in Jobs, t in
       TimeSlots2]);
8  array[Jobs] of int: consumptions;
9  array[Jobs] of int: durations;
10 array[Jobs] of int: estarts;
11 array[Jobs] of int: lends;
12 array[Jobs, TimeSlots1] of int: costs1;
13 array[Jobs, TimeSlots1] of int: bestTimesSlots1;
14 array[Scenarios, Jobs, TimeSlots2] of int: costs2;
15 array[Scenarios, Jobs, TimeSlots2] of int: bestTimesSlots2;
16 % --------- variables ----------------------------------------- %
17 array[Jobs] of var 0..numOfTimeSlots1: schedule_stage1;
18 array[Jobs] of var 0..costLimit1: actualCosts_stage1;
19 array[Scenarios,Jobs] of var 0..numOfTimeSlots2: schedule_stage2;
20 array[Scenarios,Jobs] of var 0..costLimit2: actualCosts_stage2;
21 % --------- constraints ---------------------------------------- %
22 constraint forall ( j in Jobs ) ( forall ( sc in Scenarios) (
23   (schedule_stage1[j] > 0) <-> (schedule_stage2[sc,j] = 0)));
24
25 constraint forall ( j in Jobs) ( schedule_stage1[j] > 0 ->
26   bestTimesSlots1[j,schedule_stage1[j]] >= estart[j] /\
27   bestTimesSlots1[j,schedule_stage1[j]] + durations[j] -1 <= lend[j]);
28
29 constraint forall ( sc in Scenarios ) ( forall ( j in Jobs ) (
30   schedule_stage2[sc,j] > 0 ->
31      (bestTimesSlots2[sc,j,schedule_stage2[sc,j]]+numOfTimeSlots1 >= estart[j]
32      /\  bestTimesSlots2[sc,j,schedule_stage2[sc,j]]
33          + durations[j]-1+numOfTimeSlots1 <= lend[j])));
34 constraint forall (j in Jobs) (
35  actualCosts_stage1[j] = costs1[j,schedule_stage1[j]]);
36
37 constraint forall( j in Jobs )( forall (sc in Scenarios) (
38   actualCosts_stage2[sc,j] = costs2[sc,j,schedule_stage2[sc,j]]));
39 % --------- objective ------------------------------------------ %
40 solve minimize sum ( sc in Scenarios ) ( % expected cost
41     weights[sc] * sum ( j in Jobs ) ( actualCosts_stage1[j] )
42   + weights[sc] * sum ( j in Jobs ) ( actualCosts_stage2[sc, j] ));
```

The constraints are given in lines 22-38. They correspond to the constraints from Model 4.2, with the difference that the cumulative constraint is decomposed for the stages. In the objective (line 40), we minimize the expected cost over both stages.

5.2 MIP-based stochastic Model

We show excerpts (due to space) of the stochastic MIP-based model in Fig. 4. The parameters (line 2) are the very same as in the stochastic CP model. The decision variables, however, differ: the 0-1 variables `schedule` (line 4) represent the household schedule with respect to the cheapest time slots. This means that `schedule_stage1[i,j]` is '1', if job `i` is scheduled to start at the `j`th cheapest time slot, and '0' otherwise. The reason for this is to embed the variable ordering (by cheapest timeslot) into `schedule`,

Fig. 4. Stochastic Two-stage MIP formulation of the Household Load Scheduling Problem

```
% ------ input parameters ------------------------------------- %
% same as stochastic CP model
% --------- variables ----------------------------------------- %
array[Jobs, TimeSlots1] of var 0..1: schedule_stage1;
array[Jobs] of var 0..numOfTimeSlots1: actualStarts_stage1;
array[Jobs] of var 0..costLimit1: actualCosts_stage1;
array[Scenarios, Jobs, TimeSlots2] of var 0..1: schedule_stage2;
array[Scenarios, Jobs] of var 0..costLimit2: actualCosts_stage2;
array[Scenarios, Jobs] of var 0..numOfTimeSlots2: actualStarts_stage2;
% --------- constraints ----------------------------------------- %
constraint forall(sc in Scenarios) (forall(j in Jobs) (sum(t in TimeSlots1)
  schedule_stage1[j,t] + sum(t in TimeSlots2) (schedule_stage2[sc,j,t]) == 1));

constraint forall(j in Jobs) (actualCosts_stage1[j] =
  sum(t in TimeSlots1)(schedule_stage1[j,t] * costs1[j,t]));

constraint forall(j in Jobs) (actualStarts_stage1[j] =
  sum(t in TimeSlots1)(schedule_stage1[j,t] * bestTimesSlots1[j,t]));

constraint forall(sc in Scenarios) (
  forall(j in Jobs) (actualCosts_stage2[sc,j]  =
    sum(t in TimeSlots2)(schedule_stage2[sc,j,t] * costs2[sc,j,t])));

constraint forall(sc in Scenarios) (
  forall(j in Jobs) (actualStarts_stage2[sc,j] =
    sum(t in TimeSlots2)(schedule_stage2[sc,j,t] * bestTimesSlots2[sc,j,t])));

constraint forall(j in Jobs,t in TimeSlots1) (
  schedule_stage1[j,t] == 0 \/ (bestTimesSlots1[j,t]>=estarts[j] /\
                          bestTimesSlots1[j,t]+durations[j]-1 <= lends[j]));
constraint forall(sc in Scenarios) (
  forall(j in Jobs,t in TimeSlots2) ( schedule_stage2[sc,j,t] == 0 \/
     (bestTimesSlots2[sc,j,t] + numOfTimeSlots1 >= estarts[j] /\
      bestTimesSlots2[sc,j,t] + numOfTimeSlots1 <= lends[j] - durations[j] + 1)));
% --------- objective ------------------------------------------ %
solve minimize sum ( sc in Scenarios ) ( % expected cost
    weights[sc] * sum ( j in Jobs ) ( actualCosts_stage1[j] )
  + weights[sc] * sum ( j in Jobs ) ( actualCosts_stage2[sc,j] ) );
```

since they are the search variables. In line 5, the variables `actualStart` represent the actual start times of each job, where `actualStart[j]` is the time slot at which job `j` is scheduled. Third, the variables `actualCosts` (line 6) contain the cost for each job with respect to when they are scheduled. All variables are divided by stage.

The constraints are given in line 11-34, where we first state that each job has to be scheduled once, either in the first or the second stage (line 11). Second, we link the actual costs with the schedule for the first stage (line 14) and the second stage (line 20), and set the actual start times for the first stage (line 17) and the second stage (line 24). Finally, we need to split the cumulative constraint into its decomposition for the first stage (line 28) and the second stage (line 31). Finally, we state the objective in line 36, where we minimize over the expected costs of both stages.

6 Related Work

A number of research projects have been conducted on automated demand response techniques [5, 11]. Some of them study the scheduling problem of a single household [6, 12] which does not guarantee to reduce the overall electricity cost in the network. Yu et.al [12] discuss a method that addresses the scheduling problem in a single house under the market with real time pricing. It considers the customer satisfaction by calculating the delay of each electric appliance as part of the cost. It is more suitable for a community in which customers have different electric consumption patterns.

Other approaches [2–4, 7] do not include a feedback loop from customers to service providers, which means that communications between them are one-way, and consumers cannot bargain with service providers in real time.

The method presented by van den Briel et.al. [2] predicts the shiftable loads in a day ahead, then calculates an ideal load curve that the energy generators wish to supply. The smart meter in each household stores the ideal load curve, and derives a probability distribution from it. Whenever a customer submits a request to run an electric appliance in a specific time window, the smart meter schedules the electric appliance at a random time within the specified time window, following the probability distribution. This method simplifies the communication between service providers and consumers, and minimizes the data flow in the network. This model best suits customers who do not have strict requirements on the starting time of each job.

Chen et.al. [4] use Stackelberg's competition model to calculate an optimal solution for each household. Each household is equipped with an energy schedule controller that submits an request to the service provider when the customer wants to schedule an electric appliance. Then the service provider and the energy schedule controller form a Stackelberg game to find the best time to schedule the electric appliance. The process keeps repeating whenever a new request is submitted. This method makes good use of the two-way communication network, however, customers have limited control on when the electric appliance will be scheduled.

Mohsenian-Rad et.al [7] introduce a method that requires coordination between households to reach a global optimal solution. The households comunicate with each other by broadcasting their electric appliances' schedule, and updating the electric consumption schedules when they receive the information from other houses. This process is repeated until a global optimal solution is found (and always guarantees a global optimum). The objective of this work is closer to our research, though it best suits customers who are willing to share their electric consumption data with the others.

7 Preliminary Experimental Results

The experiments were conducted on a Macbook Pro (OS X Version 10.9.4) with a 3GHz Intel Core i7 processor using 8GB 1600 MHz DDR3 RAM and a Flash storage. The data preprocessor was implemented in Python 2.7.6, using the NumPy 1.8.1 library. The deterministic models and the stochastic model were implemented in MiniZinc 2.0. The CP model was solved using Gecode; the stochastic model was solved by the G12-MIP solver.

7.1 Experiment data

For the deterministic model, the input data consists of the jobs details for each household, a dynamic pricing table, and an initial price for each time slot in a day. For the stochastic model, the input data consists of the jobs details for one household, an initial price for each time slot in stage one, and an initial price for each time slot in stage two for each scenario.

We generate test data sets from a job pool of 41 jobs that are based on real world data. For instance, the electric consumption per hour of each job comes from real data published at the AusGrid website[3]. Jobs were divided into 3 categories: jobs with limited flexibility (jobs whose care factor is 1), jobs with some flexibility (jobs whose care factor is 0.5), and jobs with full flexibility (jobs whose care factor is 0). Each experiment selects jobs from the pool to generate an instance.

The initial prices were generated from the historic aggregated electricity price for Victoria, Australia region published at AEMO[4] website. They are illustrated in Figure 5, where scenario 1 corresponds to the initial electricity price for the deterministic model.

The price lookup table is derived from a supply curve based on a year's historical data from the Australian National Energy Market (NEM) where we used the average relationship between wholesale market spot price. This is the price that changes every half hour, and is paid by all electricity retailers for all the bulk purchases they make on behalf of their retail customers. It is also the price that all power station operators earn for power generated in the same half hour. By fitting a simple curve to the historical data we were able to derive a representative supply-demand function. This function was then transformed by rescaling the demand to conform to the load levels in our model and the prices were rescaled to conform to average end-user retail prices.

This is a simple model and therefore is unavoidably distortive. In particular, it does not correctly reflect the way that the contribution of transmission and distribution network costs is incorporated into retail prices. In essence we assume that these network costs scale linearly as the generation costs increase. In reality these costs are fixed on an annual basis and amortised over expected energy demand using complex network

[3] AusGrid website: http://www.ausgrid.com.au/

[4] AEMO: Australian Energy Market Operator. http://www.aemo.com.au/

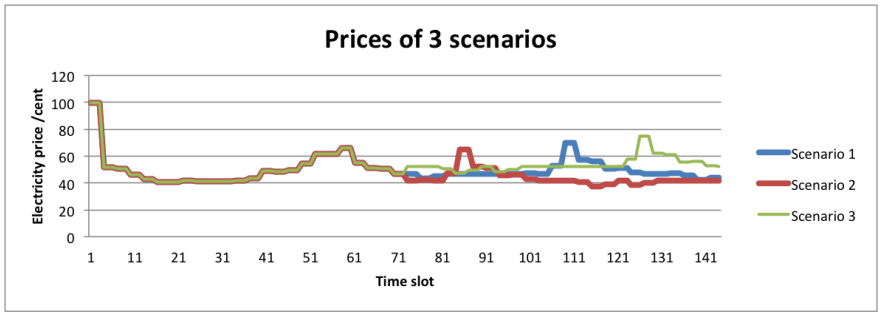


Fig. 5. Initial prices for the stochastic model; scenario 1 for the deterministic model.

Fig. 6. Run times of the deterministic model for category 1

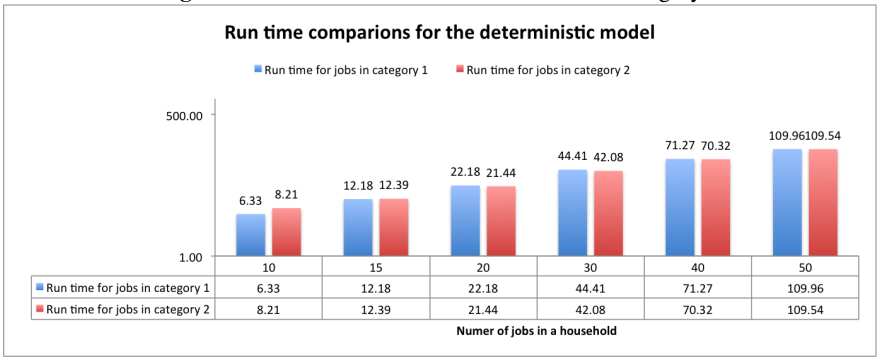


Table 3. Solutions for all data sets

Data Set	Model Type	Jobs										Total cost
		1	2	3	4	5	6	7	8	9	10	[cent]
1	CP	1	119	122	16	139	16	103	76	76	85	1391.22
2	CP	1	117	122	16	139	29	124	76	76	95	1400.92
3	CP	1	117	122	16	139	32	124	76	76	101	1408.39

tariff design models. However our model only needs to be a stylistic representation of the much more complex market and therefore is sufficient for our purposes.

7.2 Experiments for the deterministic model on a single household

We first tested the CP model on 12 data sets with 144 time slots, where each time slot has 10 minutes duration. The data sets have 10, 15, 20, 30, 40 and 50 jobs and two categories: limited flexibility (the jobs' care factor are 1) and full flexibility (the jobs' care factor is 0).

Run time comparison The experimental results for the both categories are illustrated in Fig. 6. The run time is given in milliseconds (ms) on a logarithmic scale.

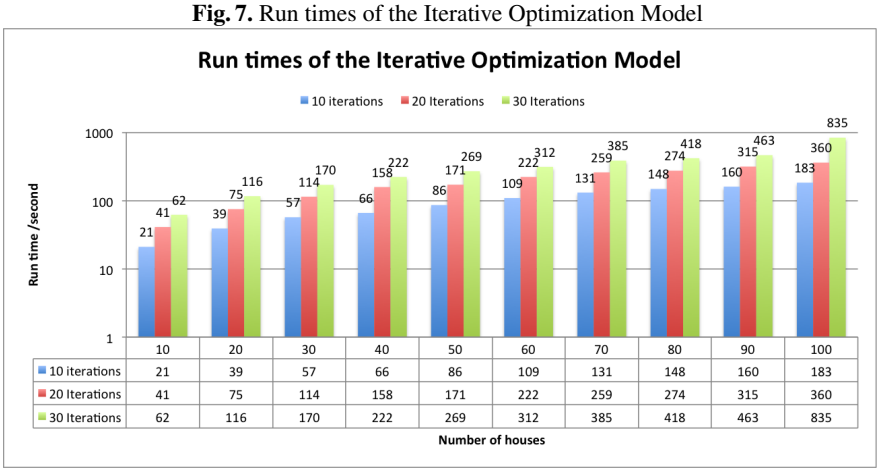
Solutions comparison To compare the solutions, the CP model was tested on 3 sets of data. The data set 1 had jobs with full flexibility; the data set 2 had jobs with some flexibility; the data set 3 had jobs with limited flexibility. The experiment results are illustrated in Table 3.

7.3 Experiments for the deterministic model on multiple households

In this experiment, we tested the run time performance, and the solutions produced by the Iterative Optimization Model.

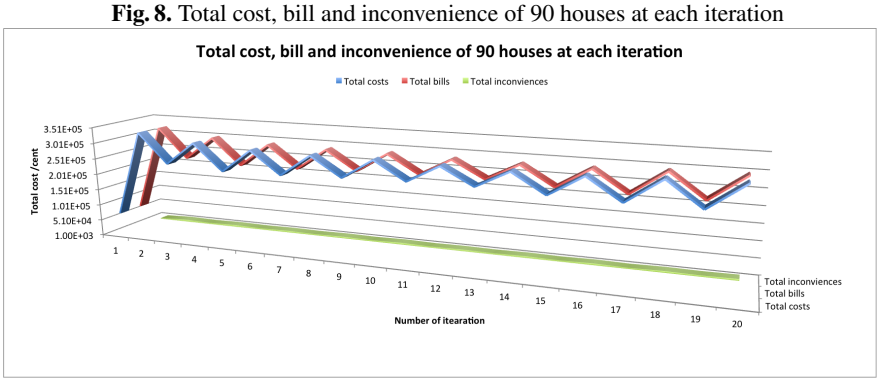
10 sets of data were generated to test the performance. Each data set had 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 houses respectively. Each house randomly selected

Fig. 7. Run times of the Iterative Optimization Model



	10	20	30	40	50	60	70	80	90	100
10 iterations	21	39	57	66	86	109	131	148	160	183
20 Iterations	41	75	114	158	171	222	259	274	315	360
30 Iterations	62	116	170	222	269	312	385	418	463	835

Fig. 8. Total cost, bill and inconvenience of 90 houses at each iteration



5 to 15 jobs from the job pool. 10 sets of data were tested on a 10, 20, and 30 iteration base. The results are illustrated in Figure 7. The run time is given in seconds (s) on a logarithmic scale. The results showed that when the number of houses is 100, the full model ran 30 iterations within 900 seconds (15 minutes).

To test the solutions of the Iterative Optimization Model, a data set consisting of 90 houses is generated. Each house randomly selected 5 to 15 jobs from the job pool. The total cost, bill and inconvenience of 90 houses in each iteration were calculated. The results are illustrated in Figure 8.

7.4 Experiments for the stochastic models

In this experiment, we tested the performances and the solutions of the stochastic models using the prices illustrated in Figure 5.

The CP stochastic model could not reach a solution within one minute time limit even on the smallest instances.The MIP stochastic model, however, could find a solution. The runtimes of the MIP stochastic model are illustrated in Figure 9 in seconds (s) on a logarithmic scale. The solutions are illustrated in Figure 10 and displays the schedules of 3 scenarios in stage 1 are the same, and the schedules in stage 2 are different.

Fig. 9. Run times of the stochastic model

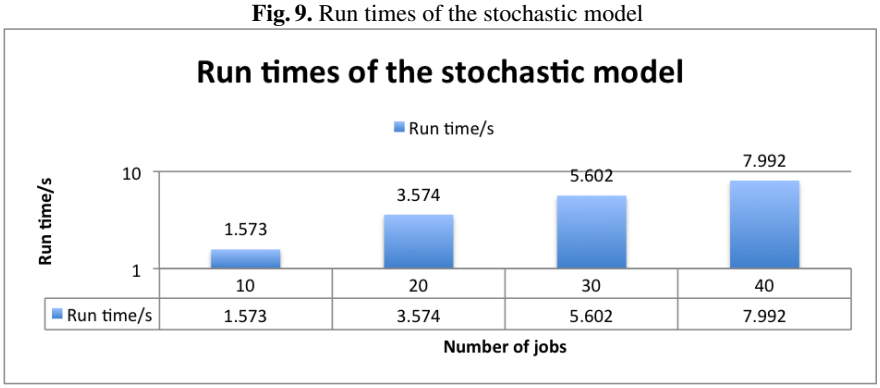
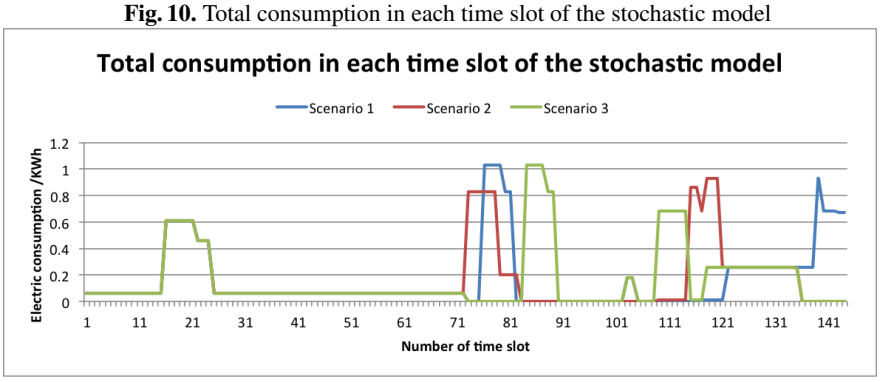


Fig. 10. Total consumption in each time slot of the stochastic model



8 Conclusions

This paper contributes to the much growing research on residential load management under real-time pricing. This work aims to minimize the overall electricity cost at a community level as well as inconvenience caused by shifting electric appliances, by providing a feedback loop from consumers to service providers, enabling them to bargain with service providers.

At the current stage, we are working towards finding a global optimal solution for customers in an acceptable time frame. In next steps, we will investigate on how to eliminate oscillations produced from the Iterative Optimization Model. We will further increase the scale of the problem to 100 and 1000 houses, and study the performance of the Iterative Optimization Model on bigger problems.

References

1. Albadi, M.H., El-Saadany, E.: Demand response in electricity markets: An overview. In: Power Engineering Society General Meeting, 2007. IEEE. pp. 1–5 (June 2007)
2. van den Briel, M., Scott, P., Thiebaux, S.: Randomized load control: A simple distributed approach for scheduling smart appliances. In: International Joint Conference on Artificial Intelligence (IJCAI). Beijing, China (August 2013)
3. Caron, S., Kesidis, G.: Incentive-based energy consumption scheduling algorithms for the smart grid. In: Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on. pp. 391 –396 (oct 2010)
4. Chen, C., Kishore, S., Snyder, L.: An innovative rtp-based residential power scheduling scheme for smart grids. In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. pp. 5956–5959 (May 2011)
5. Lee, J., Kim, H.J., Park, G.L., Kang, M.: Energy consumption scheduler for demand response systems in the smart grid. Journal of Information Science & Engineering 28(5) (2012)
6. Levorato, M., Goldsmith, A., Mitra, U.: Residential demand response using reinforcement learning. In: in Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on. pp. 409–414 (2010)
7. Mohsenian-Rad, A.H., Wong, V., Jatskevich, J., Schober, R., Leon-Garcia, A.: Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. Smart Grid, IEEE Transactions on 1(3), 320–331 (Dec 2010)
8. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) CP'07. LNCS, vol. 4741, pp. 529–543. Springer (2007)
9. Rendl, A., Tack, G., Stuckey, P.J.: Stochastic MiniZinc. In: CP'14. p. to be published. LNCS, Springer (2014)
10. Tarim, A., Manandhar, S., Walsh, T.: Stochastic Constraint Programming: A Scenario-Based Approach. Constraints 11(1), 53–80 (2006)
11. Ullah, M.N., Mahmood, A., Razzaq, S., Ilahi, M., Khan, R.D., Javaid, N.: A Survey of Different Residential Energy Consumption Controlling Techniques for Autonomous DSM in Future Smart Grid Communications. ArXiv e-prints (Jun 2013)
12. Yu, R., Yang, W., Rahardja, S.: Optimal real-time price based on a statistical demand elasticity model of electricity. In: Smart Grid Modeling and Simulation (SGMS), 2011 IEEE First International Workshop on. pp. 90–95 (Oct 2011)