Optimierte Ablaufpläne für Ressourcenverbraucher bei zeitvariablen Ressourcentarifen*

Walter Weininger, Armin Wolf
Fraunhofer FIRST, Kekuléstraße 7, 12489 Berlin
walter.weininger@first.fraunhofer.de, armin.wolf@first.fraunhofer.de

Abstract: Seit Beginn des Jahres 2011 sind deutsche Energieversorger gesetzlich verpflichtet, zukünftig zeitvariable Stromtarife anzubieten. In Irland gibt es bereits Tarife (siehe `http://www.sem-o.com`), bei denen der Strompreis sich jede halbe Stunde ändert. Außer für Strom kann es zeitvariable Tarife auch für andere Ressourcenarten geben. Die Optimierung der Ablaufpläne für Ressourcenverbraucher bei solchen Tarifen bietet Sparpotential.

Ziel dieser Arbeit ist es, mit Hilfe der Constraint-Programmierung optimale Ablaufpläne für Ressourcenverbraucher bei zeitvariablen Ressourcentarifen zu erstellen. Dazu ist das Planungsproblem als Optimierungsproblem unter Randbedingungen zu modellieren und so zu optimieren, dass die Kosten der verbrauchten Ressourcen möglichst minimal sind.

1 Einführung

Bis dato spielten zeitvariable Ressourcentarife keine große Rolle, so dass zeitabhängige Preise bei der Optimierung von Ablaufplänen in der Regel nicht berücksichtigt werden. So weisen Tarik Hadzic und Helmut Simonis in *"Creating Tests for a Family of Cost Aware Resource Constraints"* [HS10] darauf hin, dass es vor ihrer Arbeit noch keine Benchmarks für Resourcen-Planungsprobleme mit variablen Kosten gegeben hat.

Zum Beispiel können gerade zeitvariable Stromtarife Anreize zum Energiesparen und zur besseren Steuerung des Energieverbrauchs setzen (vgl. [Här11]). Das Verhalten der Nutzer kann durch die zeitabhängige Preise beeinflusst werden, um die Spitzenlasten zu vermeiden und die erneuerbaren Energienquellen effektiver auszunutzen.

In dieser Arbeit wird ein Constraint-Modell erstellt, das für die Optimierung des Ablaufplans für die zeitlich flexiblen Ressourcenverbraucher bei einem zeitvariablen Tarif über einen festgelegten Zeitraum benutzt wurde. Dieses Constraint-Modell wird für die Variablen mit endlichen ganzzahligen Domänen erstellt.

Es wird gezeigt, welche Constraints eingesetzt werden, um bestimmte Abhängigkeiten zwischen den Verbrauchern zu modellieren und wie nach einem optimalen Ablaufplan, d. h. mit minimalen Kosten, erfolgreich gesucht werden kann.

*Das zugrunde liegende Projekt wurde unter anderem mit EFRE-Mitteln der Europäischen Union gefördert.

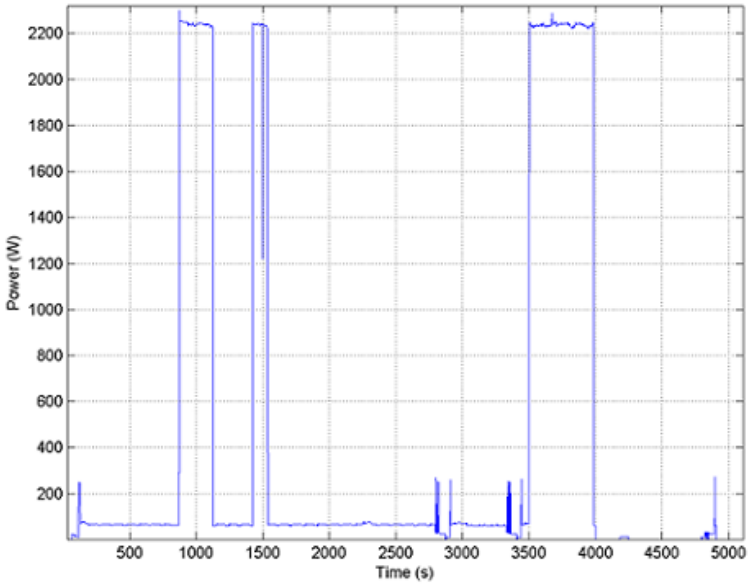


Abbildung 1: Verbrauchsprofil eines Geschirrspülers

Diese Arbeit ist wie folgt aufgebaut: nach der Einleitung wird im zweiten Teil das Constraint-Modell erklärt; im dritten Teil wird die Suchstrategie für den optimalen Ablaufplan beschrieben; im vierten Teil wird das gewählte Testszenario erläutert und die damit erzielten Testresultate werden ausgewertet; die Arbeit endet mit einem Fazit.

2 Constraint-Modell

2.1 Verbraucher und Arbeitsaufgaben

Ein zeitlich flexibler Verbraucher V wird durch Variablen für seine Startzeit $\mathsf{start}(V)$, seine Endzeit $\mathsf{end}(V)$ und seine Dauer $\mathsf{duration}(V)$ des Verbrauchs modelliert. Für jeden Verbraucher V gilt: $\mathsf{start}(V) + \mathsf{duration}(V) = \mathsf{end}(V)$.

Während der Dauer einiger Verbraucher wird die von ihnen konsumierte Ressourcenmenge nicht immer konstant sein. Solche Verbraucher können auch in den privaten Haushalten vorgefunden werden. Beispiele dafür sind Waschmaschine, Wäschetrockner, Geschirrspüler. Bei solchen Geräten werden unterschiedliche Arbeitschritte (Wasser einpumpen, Trommel drehen, Wasser abpumpen, etc) nacheinander ausgeführt und verbrauchen unterschiedlich viel Strom. Abbildung 1 zeigt den Verbrauchsprofil eines Geschirrspülers. Es ist deutlich zu sehen, dass die einzelnen Phasen während seiner Aktivität unterschiedlichen Stromverbrauch haben. Darum wird jeder Verbraucher V als einer Folge von Arbeitsauf-

gaben *(engl. tasks)* $[A_1^V, \ldots, A_k^V]$ modelliert.

Eine Arbeitsaufgabe A wird durch Variablen für ihre Startzeit $\mathsf{start}(A)$, ihre Endzeit $\mathsf{end}(A)$, ihre Dauer $\mathsf{duration}(A)$, ihren Ressourcenverbrauch $\mathsf{consumption}(A)$ und ihre Kosten $\mathsf{cost}(A)$ modelliert. Für jede Arbeitsaufgabe A gilt $\mathsf{start}(A) + \mathsf{duration}(A) = \mathsf{end}(A)$. Für die erste Arbeitsaufgabe A_1^V und letzte A_k^V eines Verbrauchers V gilt $\mathsf{start}(A_1^V) = \mathsf{start}(V)$ und $\mathsf{end}(A_k^V) = \mathsf{end}(V)$.

Im Folgenden habe jede Arbeitsaufgabe eine feste Dauer und einen festen Verbrauch. Die Arbeitsaufgaben eines Verbrauchers werden ohne Verzögerung nacheinander ausgeführt. Somit gilt für zwei aufeinanderfolgende Arbeitsaufgaben A_i^V und $A_{i+1}^V (i = 1, \ldots, k-1)$ eines Verbrauchers V: $\mathsf{end}(A_i^V) = \mathsf{start}(A_{i+1}^V)$. Außerdem gelte $\sum_{i=1}^{k} \mathsf{duration}(A_i^V) = \mathsf{duration}(V)$.

Es sei bemerkt, dass mehrere aufeinanderfolgende Arbeitsaufgaben mit gleichem Ressourcenverbrauch zu einer einzigen Arbeitsaufgabe zusammengefasst werden können. Pausen zwischen den einzelnen Arbeitsaufgaben können als Arbeitsaufgaben mit dem Verbrauch 0 modelliert werden. Ein Verbraucher kann auch nur aus einer einzigen Arbeitsaufgabe bestehen. In diesem Fall haben Verbraucher und Arbeitsaufgabe identische Startzeit, Endzeit und Dauer.

Später wird gezeigt, dass es günstig ist, Verbraucher als ganze Einheiten, d.h. nur als eine Arbeitsaufgabe, zu betrachten, obwohl sie eigentlich eine Abfolge der Arbeitsaufgaben darstellen.

2.2 Randbedingungen (Constraints)

Neben den bereits geltenden zeitlichen Bedingungen sind bei der Planung der zeitlichen Abfolge von Verbrauchern (Ablaufplanung) weitere, im Wesentlichen durch den Nutzer festgelegte Einschränkungen und Bedingungen zu berücksichtigen. In diesem Abschnitt wird daher gezeigt, welche Einschränkungen wie modellierbar sind. Das Betrachten eines Verbrauchers wiederum als Arbeitsaufgabe *(engl. task)* mit Start- und Endezeit (vgl. Abschnitt 2.1) vereinfacht die Modellierung einiger dieser Einschränkungen.

Die Einschränkungen, sowie die Tarif- und Verbraucherdaten, werden im XML-Format beschrieben. Die XML-Instanz mit diesen Daten wird dann zum Konstruieren des Constraint-Modells benutzt. Die ermittelten Startzeiten und Kosten für die optimale Konfiguration werden auch in das XML-Format exportiert und können weiterverwendet werden.

2.2.1 Sperrzeiten

Soll ein Verbraucher V zu bestimmten Zeiten nicht aktiv (planbar) sein, kann diese Bedingung durch geeignete Wahl der Wertebereiche (Domänen) seiner Start- und Endzeitvariablen festgelegt werden. Im Folgenden wird daher mit $D(X)$ die *Domäne* einer Variablen X bezeichnet.

Sind für den Verbraucher V Sperrzeitintervalle definiert, d.h. Intervalle, mit denen dieser

Verbraucher sich niemals überschneiden darf, dann gilt für jedes Sperrzeitintervall $[a, b]$ und jeden Zeitpunkt t:

- wenn $t \in [a + 1 - \mathsf{duration}(V), b - 1]$, dann $t \notin D(\mathsf{start}(V))$,
- wenn $t \in [a + 1, b - 1 + \mathsf{duration}(V)]$, dann $t \notin D(\mathsf{end}(V))$.

Die Grenzen der Intervalle, $a - \mathsf{duration}(V)$ und b bzw. a und $b + \mathsf{duration}(V)$, werden aus den Domänen der Variablen nicht entfernt, denn ein Verbraucher kann im Zeitpunkt a enden oder im Zeitpunkt b starten.

Durch das Berücksichtigen der Dauer des Verbrauchers vor und nach dem Sperrzeitintervall wird sichergestellt, dass der Verbraucher sich mit diesem Sperrzeitintervall niemals überschneiden wird.

Die Sperrzeiten stellen "Löcher" in den Domänen der Start- und Endzeiten von Verbrauchern und Arbeitsaufgaben dar. Die entsprechenden Variablen werden durch die Summenconstraints miteinander verbunden. Für diese Constraints kann die Grenzen- oder Domänenkonsistenz hergestellt werden.

Bei der Grenzenkonsistenz werden nur die Grenzen der Variablendomänen angepasst, ohne die inkonsistenten Werte dazwischen auszuschließen. Bei der Domänenkonsistenz werden auch alle inkonsistente Werte innerhalb der Domänengrenzen bei der Propagation ausgeschlossen.

Für die Domänen mit "Löchern" ist die Herstellung der Domänenkonsistenz in einem Summenconstraint aufwendiger als die Herstellung der Grenzenkonsistenz, dafür aber können eventuell mehr inkonsistente Werte ausgeschlossen werden. Wie sich diese beiden Konsistenzarten auf die Rechenzeit auswirken, wird im Abschnitt 4.2 untersucht.

2.2.2 Kapazitätsbeschränkung

Der Gesamtverbrauch aller Arbeitsaufgaben $\{A_1, \ldots, A_m\}$ darf zu keinem Zeitpunkt einen bestimmten Grenzwert limit überschreiten. Diese Bedingung wird mit Hilfe des `Cumulative`-Constraints [SW10, SWS05, UWB+06] formuliert: $\texttt{Cumulative}(\{A_1, \ldots, A_m\}, \mathsf{limit})$, dessen Semantik den genannten Sachverhalt widerspiegelt:

$$\forall t \sum_{A \in \{A_1,\ldots,A_m\} | \mathsf{start}(A) \leq t < \mathsf{end}(A)} \mathsf{consumption}(A) \leq \mathsf{limit} \ .$$

Bei der Betrachtung elektrischer Verbraucher in Haushalten ist eine solche kapazitive Beschränkung durch die Hauptsicherung gegeben.

2.2.3 Nicht-Überlappung

Haushaltstypische Verbraucher, wie z. B. Waschen, benötigen im Allgemeinen die (einzige) Waschmaschine exklusiv. Es soll daher möglich sein, für zwei unterschiedliche Ver-

braucher V_i und V_j festzulegen, dass diese niemals gleichzeitig aktiv sein dürfen. Diese Bedingung wird durch eine Disjunktion zweier Ungleichungen ausgedrückt:

$$(\mathsf{end}(V_i) \leq \mathsf{start}(V_j)) \quad \vee \quad (\mathsf{end}(V_j) \leq \mathsf{start}(V_i)) \ .$$

Sind es mehrere Verbraucher $V_1, \ldots, V_n$, die das selbe Gerät exklusiv benötigen, so entsprechen die Disjunktionen dem Serialisieren der Aktivitäten der Verbraucher auf einer exklusiven Ressource. Dies kann mit Hilfe eines `SingleResource`-Constraints mittels `SingleResource`$(\{V_1, \ldots, V_n\})$ modelliert werden. Dies bietet den Vorteil, dass für dieses *globale* Constraint effiziente suchraumeinschränkende Verfahren eingesetzt werden können (vgl. [Vil04]).

Idealerweise werden solche Constraints für die Verbraucher, die die Aktivierungen eines exklusiven Geräts beschreiben, automatisch generiert, wenn die Zeitfenster für deren mögliche Aktivierungen sich überlappen. Dies wird bei unserer Umsetzung der gewählten Spezifikationssprache in XML geleistet.

2.2.4 Ablaufreihenfolge

Ein Verbraucher V_2 darf nur nach dem Verbraucher V_1 aktiv sein. Das wird durch das Summenconstraint ausgedrückt:

$$\mathsf{end}(V_1) + \mathsf{delay}(V_1, V_2) = \mathsf{start}(V_2) \ .$$

Dabei ist $\mathsf{delay}(V_1, V_2)$ eine Variable, mit der eine zulässige Verzögerung zwischen V_1 und V_2 modelliert wird. $D(\mathsf{delay}(V_1, V_2))$ enthält ursprünglich alle zulässigen Verzögerungen zwischen diesen beiden Verbrauchern.

Wenn für einen oder für beide Verbraucher Sperrzeiten vorgegeben sind, dann wird durch das Herstellen der Domänenkonsistenz (vgl. Abschnitt 2.2.1) in diesem Constraint erreicht, dass die Sperrzeiten eines Verbrauchers sich auf die Domänen der Variablen des anderen Verbrauchers auswirken.

Zwei Sonderfälle können anders ausgedrückt werden. Wenn es keine Verzögerung geben darf, dann gelte $\mathsf{end}(V_1) = \mathsf{start}(V_2)$. Wenn es keine Einschränkung für die Verzögerung gibt, dann gelte $\mathsf{end}(V_1) \leq \mathsf{start}(V_2)$.

2.2.5 Aktivierungen in einer Schicht

Wenn der gesamte Zeitraum in mehrere gleich lange Schichten aufgeteilt ist, so kann die Bedingung modelliert werden, dass mehrere bestimmte Verbraucher in der gleichen Schicht starten oder enden sollen.

Sei die Anzahl der Schichten PERIODS und die Dauer jeder Schicht SHIFTDURATION. Der Zeitraum beginnt im Zeitpunkt T_0. PERIODS, SHIFTDURATION und T_0 sind Konstanten. Jede Schicht ist von 0 bis PERIODS $- 1$ nummeriert und die erste Schicht beginnt im Zeitpunkt T_0. Weiterhin werden noch folgende Hilfsvaribalen benötigt:

- $\mathsf{shiftnumber}(V)$ – Nummer der Schicht, in der der Verbraucher V startet oder endet; es gilt: wenn $x \in D(\mathsf{shiftnumber}(V))$, dann $x \in [0, \mathsf{PERIODS} - 1]$;
- $\mathsf{shiftstart}(V)$ – Startzeit des Verbrauchers V seit dem Beginn der Schicht; es gilt: wenn $x \in D(\mathsf{shiftstart}(V))$, dann $x \in [0, \mathsf{SHIFTDURATION} - 1]$;
- $\mathsf{shiftend}(V)$ – Endzeit des Verbrauchers V seit dem Beginn der Schicht; es gilt: wenn $x \in D(\mathsf{shiftend}(V))$, dann $x \in [1, \mathsf{SHIFTDURATION}]$.

Die Start- und Endzeiten der Verbraucher können aus diesen Hilfsvariablen berechnet werden:

- $T_0 + \mathsf{shiftnumber}(V) * \mathsf{SHIFTDURATION} + \mathsf{shiftstart}(V) = \mathsf{start}(V)$,
- $T_0 + \mathsf{shiftnumber}(V) * \mathsf{SHIFTDURATION} + \mathsf{shiftend}(V) = \mathsf{end}(V)$.

Sollen mehrere Verbraucher $\{V_1, \ldots, V_n\}$ in der gleichen Schicht starten, so werden folgende Constraints benötigt, um diese Bedingung zu modellieren:

$$T_0 + \mathsf{shiftnumber} * \mathsf{SHIFTDURATION} + \mathsf{shiftstart}(V_i) = \mathsf{start}(V_i)$$

für jeden Verbraucher $V_i \in \{V_1, \ldots, V_n\}$ mit der gleichen Variablen shiftnumber für alle $i (i = 1, \ldots, n)$.

Sollen diese Verbraucher in der gleichen Schicht enden, so werden statt $\mathsf{shiftstart}(V_i)$ und $\mathsf{start}(V_i)$ entsprechend $\mathsf{shiftend}(V_i)$ und $\mathsf{end}(V_i)$ benutzt.

Für einen Verbraucher V_i mit $\mathsf{duration}(V_i) \leq \mathsf{SHIFTDURATION}$ kann die Bedingung modelliert werden, dass V_i innerhalb einer Schicht sowohl startet als auch endet. Dazu müssen einfach die Constraints für das Starten und für das Enden in einer Schicht mit der gleichen Variablen shiftnumber benutzt werden.

Die Domänen von shiftnumber und $\mathsf{shiftstart}(V_i)$ bzw. $\mathsf{shiftend}(V_i)$ sind so initialisiert, dass es eine eindeutige das Constraint erfüllende Belegung von diesen Hilfsvariablen für jeden Wert von $\mathsf{start}(V_i)$ bzw. $\mathsf{end}(V_i)$ gibt.

2.2.6 Aktivierungen in unterschiedlichen Schichten

Es ist auch möglich, die Bedingung zu modellieren, dass mehrere Verbraucher gerade nicht in der gleichen Schicht starten oder enden dürfen. In diesem Fall wird für jeden Verbraucher seine eigene shiftnumber-Variable benötigt – $\mathsf{shiftnumber}(V_i)$. Alle diese Variablen müssen dann unterschiedlich sein. Das kann durch das `AllDifferent`-Constraint [LOQTvB03] ausgedrückt werden:

$$\texttt{AllDifferent}(\{\mathsf{shiftnumber}(V_1), \ldots, \mathsf{shiftnumber}(V_n)\}) \ .$$

Wenn dabei mehr Verbraucher beteiligt sind, als es Schichten gibt, dann kann dieses Constraint natürlich niemals erfüllt werden.

2.3 Kostenberechnung

Für die Kostenberechnung wird nur ein zeitvariabler Tarif verwendet. Ein zeit- und mengenvariabler Tarif wird nicht betrachtet. Das bedeutet, der Preis pro verbrauchter Ressourceneinheit ist lediglich abhängig von dem Zeitraum, in dem die Ressource konsumiert wird, jedoch nicht von der insgesamt verbrauchten Menge an Ressourceneinheiten.

2.3.1 Kosten der Arbeitsaufgaben

In *"A Resource Cost Aware Cumulative"* [SH11] wurden von Helmut Simonis und Tarik Hadzic einige Ansätze zur Kostenoptimierung bei zeitvariablen Tarifen vorgestellt. Der Ansatz, der sich für die Constraint-Programmierung mit ganzzahligen Domänenwerten eignet, besteht darin, für jede Arbeitsaufgabe A ein `Element`-Constraint zu erzeugen:

$$\texttt{Element}(\mathsf{cost}(A), [C_0^A, \ldots, C_{T-\mathsf{duration}(A)}^A], \mathsf{start}(A)) \ .$$

Dabei ist T die Dauer des betrachteten Planungshorizonts.[1] $[C_0^A, \ldots, C_{T-\mathsf{duration}(A)}^A]$ sind Konstanten. $C_i^A \in [C_0^A, \ldots, C_{T-\mathsf{duration}(A)}^A]$ sind die Kosten der Arbeitsaufgabe A, wenn sie zum Zeitpunkt i $(i = 0, \ldots, T - \mathsf{duration}(A))$ starten würde. Weiterhin sei $\mathsf{Tariff}(t)$ eine Funktion, die jedem Zeitpunkt t einen Preis zuordnet, d.h. $\mathsf{Tariff}(t)$ sind die Kosten für den Verbrauch einer Ressourceneinheit im Zeitintervall $[t, t+1]$. Es gilt:

$$C_i^A = \left(\sum_{j=i}^{i+\mathsf{duration}(A)-1} \mathsf{Tariff}(j) \right) * \mathsf{consumption}(A) \ .$$

Die Werte C_i^A werden für alle Arbeitsaufgaben für alle möglichen Startzeiten im Voraus berechnet. Da die Dauer und der Verbrauch festgelegt sind, muss für jeden Zeitpunkt i nur ein Wert berechnet werden.

Das `Element`-Constraint ist erfüllt, wenn $\mathsf{cost}(A) = C_{\mathsf{start}(A)}^A$ gilt. Die Kosten aller Arbeitsaufgaben $\{A_1, \ldots, A_m\}$ werden zu Gesamtkosten totalcost summiert:

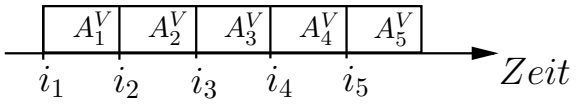
$$\sum_{i=1}^{m} \mathsf{cost}(A_i) = \mathsf{totalcost} \ .$$

Es gilt, den Wert von totalcost zu minimieren.

2.3.2 Redundante Constraints

Der Vorteil des `Element`-Constraints ist, dass sowohl die Kosten in Abhängigkeit von Startzeit als auch die Startzeit in Abhängigkeit von Kosten propagiert werden. Der Nach-

[1] Der Einfachheit halber beginnt dieser zum Zeitpunkt 0.



$$
\begin{aligned}
i_1 &= i \\
i_2 &= i + \mathsf{duration}(A_1^V) \\
&\vdots \\
i_5 &= i + \textstyle\sum_{j=1}^{4} \mathsf{duration}(A_j^V) \\
C_i^V &= C_{i_1}^{A_1^V} + C_{i_2}^{A_2^V} + C_{i_3}^{A_3^V} + C_{i_4}^{A_4^V} + C_{i_5}^{A_5^V}
\end{aligned}
$$

Abbildung 2: Kostenberechnung zum Zeitpunkt i für einen Verbraucher mit fünf Arbeitsaufgaben.

teil ist, dass dieses Constraint nur die Domänen der Variablen einer einzelnen Arbeitsaufgabe einschränkt. Es werden weder die globalen Einschränkungen (wie z.B. Kapazitätsbeschränkung) noch die Variablen (Kosten und Startzeit) anderer Arbeitsaufgaben berücksichtigt. Dieser Nachteil kann durch die redundanten Constraints abgeschwächt werden.

Die festgelegten Dauer und Verbrauch von allen Arbeitsaufgaben erlauben es, ein `Element`-Constraint auch für jeden Verbraucher V, der aus mehreren Arbeitsaufgaben besteht, zu erzeugen:

$$\texttt{Element}(\mathsf{cost}(V), [C_0^V, \ldots, C^V_{T-\mathsf{duration}(V)}], \mathsf{start}(V)) \ .$$

$C_i^V \in [C_0^V, \ldots, C^V_{T-\mathsf{duration}(V)}]$ sind die Kosten des Verbrauchers V, wenn er zum Zeitpunkt i $(i = 0, \ldots, T - \mathsf{duration}(V))$ starten würde.

Für einen Verbraucher V mit Arbeitsaufgaben $\{A_1^V, \ldots, A_k^V\}$ werden diese Konstanten wie folgt berechnet:

$$C_i^V = \sum_{q=1}^{k} C^{A_q^V}_{i+\left(\sum_{j=1}^{q} \mathsf{duration}(A_j^V)\right)-\mathsf{duration}(A_q^V)} \ .$$

In Abbildung 2 wird anhand eines Beispiels genauer erläutert, wie die Kosten eines Verbrauchers sich aus den Kosten der einzelnen Arbeitsaufgaben zusammensetzen.

Die Summe der Kosten von allen Verbrauchern $\{V_1, \ldots, V_n\}$ ist die gleiche wie die Summe der Kosten von allen Arbeitsaufgaben $\{A_1, \ldots, A_m\}$:

$$\sum_{i=1}^{m} \mathsf{cost}(A_i) = \sum_{j=1}^{n} \mathsf{cost}(V_j) = \mathsf{totalcost} \ .$$

Die Element-Constraints für Verbraucher und die Summe, die die Verbraucherkosten zu totalcost summiert, sind redundante Constraints, die jedoch helfen können, den Suchraum

effektiver einzuschränken. Das ist ein weiterer Grund, warum Verbraucher als ganze Einheiten betrachtet werden.

3 Suchstrategie

Bei der Suche nach einer Lösung ist es ausreichend, nur die Variablen für die Startzeiten der Verbraucher zu belegen. Die Belegungen von allen anderen Variablen (Endzeiten und Kosten der Verbaucher, Startzeiten, Endzeiten und Kosten der einzelnen Arbeitsaufgaben, Verzögerungen zwischen den Verbrauchern, sowie die Hilfsvariablen shiftnumber, shiftstart und shiftend) ergeben sich aus den impliziten Abhängigkeiten von den Startzeiten der Verbraucher. Es muss also zwischen den Startzeiten der Verbraucher und den Startzeiten der Arbeitsaufgaben unterschieden werden.

Zur Optimierung wird die *Branch&Bound-Methode* angewandt: Um einen Ablaufplan mit minimalen Kosten zu finden, wird zuerst eine beliebige gültige Belegung für die Startzeiten der Verbraucher gefunden. Diese Belegung liefert die Oberschranke für die Variable totalcost. Dann wird diese Variable mit immer kleineren Werten belegt und es wird versucht, gültige Belegungen von Startzeiten dafür zu finden. Das wird so lange wiederholt, bis die kleinstmögliche Belegung für totalcost gefunden wird. Genauso kann nach einem Ablaufplan mit maximalen Kosten gesucht werden. In diesem Fall müssen für die Variable totalcost immer größere Werte ausprobiert werden. Die Domäne von totalcost hat wie alle Variablen im Constraint-Modell eine endliche Domäne mit ganzzahligen Werten. Darum wird die Suche nach einem minimalen oder maximalen Wert für totalcost nach endlichen vielen Iterationen terminieren.

Die Werte aus der Domäne $D(\text{totalcost})$ können absteigend oder aufsteigend mit der konstanten Schrittweite ausprobiert werden. Das wäre monotones Optimieren. Als effektivere Suchstrategie erweist sich oft aber das dichotome Optimieren: angefangen wird mit einer großen Schrittweite, die nach jeder Verbesserung von totalcost halbiert wird. Diese beiden Strategien werden im Abschnitt 4.2 miteinander verglichen.

Die zu belegenden Variablen können nach bestimmten Kriterien sortiert werden. Im getesteten Beispiel hat es sich am effektivsten erwiesen, nach der ersten Propagation vor der Suche die Verbraucher absteigend nach der Domänenkardinalität ihrer Kostenvariablen zu sortieren, und deren Startzeiten in dieser Reihenfolge zu belegen.

4 Testresultate

4.1 Testbeispiel

Für das Testen wurde der Zeitraum von einer Woche genommen. Für das Tarif wurden die Strompreise von sem-o (siehe `http://www.sem-o.com`) aus der Woche vom 13. bis zum 19. Dezember 2010 genommen. Bei diesem Tarif können sich die Preise jede halbe

Stunde ändern und liegen im Bereich von 4 bis 40 cent/KWh. Jeder Tag wurde als eine Schicht betrachtet. Eine Zeiteinheit entsprach einer halben Stunde, somit wurde der ganze Zeitraum in 336 Zeiteinheiten aufgeteilt.

Als Stromverbraucher dienten haushaltstypische Geräte. Ihr Stomverbrauch wurde gemessen und ein Durchschnittsverbrauch für jede halbe Stunde ihrer Aktivität berechnet.

Die genauere Erfassung der Verbrauchsprofile ist aus zwei Gründen nicht sinnvoll. Zum einen würde der Zeitraum dann aus mehr Zeiteinheiten bestehen, was zur Folge einen viel größeren Suchraum und längere Rechenzeiten hatte. Zum anderen sind die Preise bis auf Hundertstel eines Cents genau und da der Verbrauch nicht sehr groß ist, wirken sich die Nachkommastellen bei den Preisen kaum oder gar nicht auf das Endergebnis aus.

Es wurde folgendes Szenario für 8 Stromverbraucher mit entsprechenden Einschränkungen und erlaubten Zeitfenstern für die Aktivierungen getestet:

- Wäschewaschen: zwei Mal in der Woche, Intensiv- und Sensitiv-Wäsche, an verschiedenen Tagen, Mo-Fr, 10-18 Uhr, je 2 Stunden;
- Wäschetrocknen: zwei Mal in der Woche, nach dem Wäschewaschen, spätestens halbe Stunde danach, je 2,5 Stunden;
- Bügeln: einmal in der Woche, nach beiden Trocknungen, frühestens eine Stunde danach, Mo-Fr, 10-18 Uhr, 1 Stunde;
- Staubsaugen: einmal in der Woche, Mo-Fr, 12-16 Uhr, 1 Stunde;
- DVD-Schauen: zwei Mal in der Woche, an verschiedenen Tagen, Mo-So, 16-22 Uhr, je 2 Stunden;
- Computern: nach dem DVD-Schauen, aber am selben Tag, spätestens 4 Stunden danach, je 2 Stunden;
- Geschirrspülen: täglich, Mo-So, 16-22 Uhr, 1,5 Stunden;
- Backen: Sa oder So, 10-20 Uhr, 4 Stunden.

Hinzu kommen noch folgende Einschränkungen:

- verschiedene Tage für Bügeln und DVD-Schauen;
- Nicht-Überlappung für DVD-Schauen, Computern, Bügeln, Staubsaugen, Backen;
- Bügeln und Staubsaugen am gleichen Tag;
- Der Gesamtverbrauch darf zu keinem Zeitpunkt den Wert von 2 KW überschreiten.

Das Limit von 2 KW wurde so gewählt, um einen Spitzenverbrauch insbesondere zu günstigen Zeiten zu vermeiden, was bei einem verbrauchsabhängigen Strompreis zu einer Preissteigerung führen würde.

ISBN 978-3-88579-286-4

Weiterhin müssen implizite Nicht-Überlappung-Bedingungen für die Aktivitäten des gleichen Stromverbrauchers beachtet werden. Diese Bedingungen werden explizit im Constraint-Modell für Waschmaschine, Wäschetrockner, DVD-Player, Computer und Geschirrspüler einbezogen. Beim Geschirrspüler ist diese Bedingung bereits durch die vorgegebenen Zeitfenster der Aktivierungen erfüllt.

4.2 Auswertung

Getestet wurde mit dem Constraint-Solver `firstcs` [Wol06] auf einem Intel Xeon CPU mit Taktrate von 2,53GHz und Arbeitsspeicher von 6GB. Tabelle 1 zeigt, wie lange es bei verschieden Konfigurationen gedauert hat, eine Belegung mit minimalen Kosten zu finden, und wie viele Rücksprünge während der Suche dabei ausgeführt wurden. Die letzte Zeile in der Tabelle zeigt die Ergebnisse für das konstruierte Modell. In diesem Modell wurden verwendet:

- domänenkonsistente Summenconstraints,
- reduntante Constraints,
- dichotomische Suchstrategie,
- Sortierung der Kostenvariablen nach absteigender Domänengröße bei der Suche.

Das so konstruierte Problem-Modell und die so gewählte Suchstrategie führten im Vergleich zum besten Ergebnis.

	Zeit in ms	Rücksprünge
Grenzenkonsistenz	> 600000	-
Monotones Optimieren	168486	3035
Ohne Sortieren	53197	20462
Ohne redundante Constraints	26757	4261
Sortiert nach Startzeiten	2055	224
Ohne Element-Constraints für jede Arbeitsaufgabe	1036	114
Konstruiertes Constraint-Modell mit spezieller Suchstrategie	1014	112

Tabelle 1: Testergebnisse für die Suche nach einer Konfiguration mit minimalen Kosten

In der ersten Zeile sieht man, dass eine Lösung auch nach 10 Minuten nicht gefunden wird, wenn auf die Domänenkonsistenz in den Summenconstraints verzichtet wird. Aus der Tabelle ist ersichtlich, dass durch das Herstellen der Domänenkonsistenz auch der größte Performanz-Gewinn erreicht wird. Das wird vor allem durch die Summenconstraints erreicht, die die Ablaufreihenfolge der Verbraucher modellieren. Und zwar dann, wenn die Verbraucher Sperrzeiten haben und die maximal erlaubte Verzögerung nicht sehr groß ist. Denn, wie im Abschnitt 2.2.4 erwähnt, werden dadurch die Sperrzeiten eines Verbrauchers auf den anderen projiziert, sodass viele inkonsistente Werte schon vor der Suche

noch während der ersten Propagation aus den Variablendomänen entfernt werden und die Reihenfolge der sortierten Variablen sich dementsprechend ändert.

Beim monotonen Optimieren hat die Suche sehr lange gebraucht. Das dichotome Optimieren im konstruierten Modell verursacht eine viel schnellere Konvergenz.

Wenn die zu belegenden Variablen nicht sortiert wurden, dann ist die größte (ausgenommen die Konfiguration mit der Grenzenkonsistenz) Anzahl von Rücksprüngen notwendig, bis die optimale Lösung gefunden ist. Durch das Sortieren wird erreicht, dass die inkonsistenten Abzweigungen im Suchbaum früher erkannt werden, was wiederum die Suche beschleunigt. Im Modell ohne redundante Constraints hat die Suche relativ lange gedauert und es waren mehr als viertausend Rücksprünge notwendig. Die redundanten Constraints propagieren die Kosten gleich für die ganzen Abfolgen von Arbeitsaufgaben, wodurch eine bessere Suchraumeinschränkung erreicht wird. Da der Tarif eine nicht injektive Funktion ist, können die Kardinalitäten der Startzeit- und der zugehörigen Kostenvariablen sich unterscheiden. Wurden die Startzeitvariablen nach ihrer Domänenkardinalität sortiert, so hat die Suche etwas länger gedauert und es waren mehr Rücksprünge notwendig. Das liegt daran, dass die Gesamtsumme der Kosten beschränkt ist, während für die Startzeiten das nicht der Fall ist.

Die Kosten für die Verbraucher können auch nur durch die redundanten Constraints berechnet werden, d.h. ohne `Element`-Constraints für jede einzelne Arbeitsaufgabe. Bei dem getesteten Szenario dauerte die Suche ohne diese `Element`-Constraints nur 22 ms länger und brauchte nur 2 Rücksprünge mehr. Bei einem größeren Anwendungsfall mit mehr Verbrauchern kann sich diese Konfiguration durchaus negativ auf die Performanz auswirken. Das Benutzen von reduntanten Summenconstraints $\mathsf{start}(V) + \mathsf{duration}(V) = \mathsf{end}(V)$ für jeden Verbraucher V, der sich aus mehreren Arbeitsaufgaben zusammensetzt, hat keinen nennenswerten Effekt, da die Start- und Endzeiten der Verbraucher schon durch die Summenconstraints der Arbeitsaufgaben propagiert werden.

5 Fazit

Im getesteten Szenario wurden typische Stromverbraucher verwendet, die in einem privaten Haushalt zu finden sind, denn der getestete Ablaufplan-Optimierer soll auf einem Web-Portal zum Einsatz kommen, so dass jeder Haushalt, sofern er einen Stromtarif mit tageszeitabhängigen Preisen hat, damit sein individuelles Sparpotential ermitteln kann. Somit ist das präsentierte Ergebnis anwendungsreif.

Die Abbildungen 3 und 4 zeigen die visualisierten Ablaufpläne mit den minimalen und maximalen Kosten für das Testszenario. Es lässt sich feststellen, dass der Unterschied zwischen den minimal möglichen Kosten von 97,22 cent und maximal möglichen Kosten von 211,50 cent mehr als 100% beträgt. Die jährlichen Kosten sind das 52-fache der wöchentlichen Kosten. Da der Tarif und der Verbrauch sich jede Woche ändern können, sind die jährlichen Kosten nur ein Schätzwert, der aber eine Vorstellung für das Sparpotential gibt. Bei dem getesteten Beispiel ist der Unterschied zwischen jährlichen minimalen und maximalen Kosten 59,43 €, was für einen privaten Haushalt ein relativ großer Be-

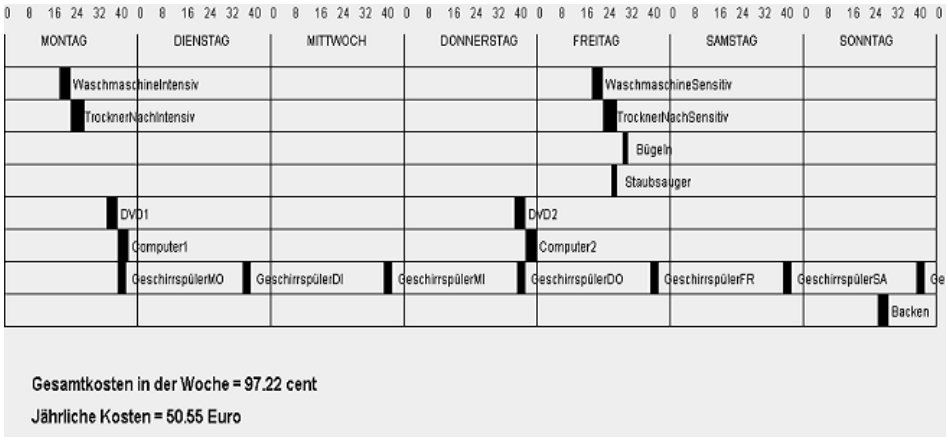


Abbildung 3: Ablaufplan mit minimalen Kosten

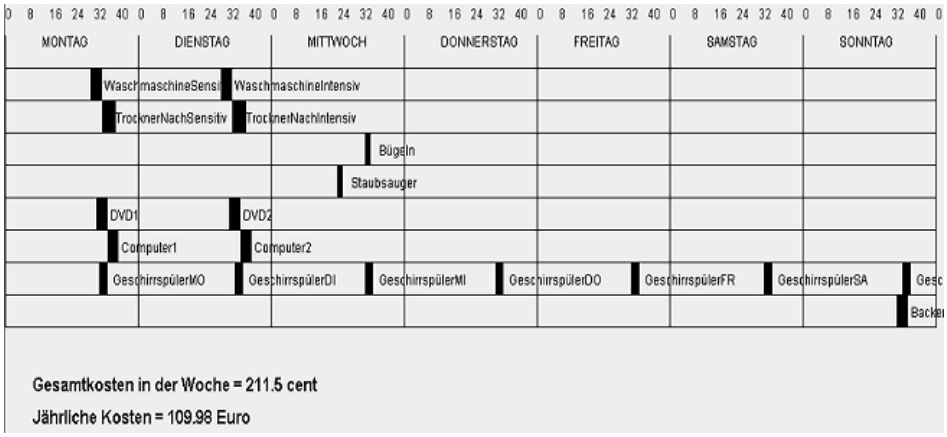


Abbildung 4: Ablaufplan mit maximalen Kosten

trag ist. Beim Übertragen dieser Erkenntnisse auf die industriellen Verhältnisse lässt sich sagen, dass die Ablaufplan-Optimierung bei einem zeitvariablen Ressourcentarif ein enormes Sparpotential bieten kann.

Literatur

[Här11] Hendrik Härter. Stromsparen mit Smart Metering stößt auf große Akzeptanz. *Elektronik Praxis*, 4. Juli 2011.

[HS10] Tarik Hadzic und Helmut Simonis. Creating Tests for a Family of Cost Aware Resource Constraints. In Barry O'Sullivan und Armin Wolf, Hrsg., *Proceedings of the ERCIM Workshop on Constraint Solving and Constraint Logic Programming*, Seiten 58–73. Fraunhofer FIRST, Berlin, Germany, 2010.

[LOQTvB03] Alejandro López-Ortiz, Claude-Guy Quimper, John Tromp und Peter van Beek. A Fast and Simple Algorithm for Bounds Consistency of the AllDifferent Constraint. In Georg Gottlob und Toby Walsh, Hrsg., *IJCAI*, Seiten 245–250. Morgan Kaufmann, 2003.

[SH11] Helmut Simonis und Tarik Hadzic. A resource cost aware cumulative. In Javier Larrosa und Barry O'Sullivan, Hrsg., *Proceedings of the 14th Annual ERCIM international conference on Constraint solving and constraint logic programming*, CSCLP'09, Seiten 76–89, Berlin, Heidelberg, 2011. Springer-Verlag.

[SW10] Andreas Schutt und Armin Wolf. A New $O(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In David Cohen, Hrsg., *CP*, Jgg. 6308 of *Lecture Notes in Computer Science*, Seiten 445–459. Springer, 2010.

[SWS05] Andreas Schutt, Armin Wolf und Gunnar Schrader. Not-First and Not-Last Detection for Cumulative Scheduling in $O(n^3 \log n)$. In Umeda et al. [UWB$^+$06], Seiten 66–80.

[UWB$^+$06] Masanobu Umeda, Armin Wolf, Oskar Bartenstein, Ulrich Geske, Dietmar Seipel und Osamu Takata, Hrsg. *Declarative Programming for Knowledge Management, 16th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2005, Fukuoka, Japan, October 22-24, 2005, Revised Selected Papers*, Jgg. 4369 of *Lecture Notes in Computer Science*. Springer, 2006.

[Vil04] Petr Vilím. $O(n \log n)$ Filtering Algorithms for Unary Resource Constraint. In Jean-Charles Régin und Michel Rueher, Hrsg., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004, Proceedings*, Jgg. 3011 of *LNCS*, Seiten 335–347, Nice, France, April 20-22 2004. SV.

[Wol06] Armin Wolf. Object-Oriented Constraint Programming in Java Using the Library firstcs. In Michael Fink, Hans Tompits und Stefan Woltran, Hrsg., *WLP*, Jgg. 1843-06-02 of *INFSYS Research Report*, Seiten 21–32. Technische Universität Wien, Austria, 2006.