Constraint Programming for Hierarchical Resource Allocation

Alexander Schiendorfer

Institute for Software & Systems Engineering, University of Augsburg
{alexander.schiendorfer}@informatik.uni-augsburg.de

Abstract. Resource allocation in a feasible or even cost-effective manner is a prominent task appearing in various domains such as energy management, grid computing, distributed heat management or the like. The mere size of future systems calls for mechanisms and algorithms that are able to address uncertainty and scalability – two issues that Organic Computing traditionally focuses on.
In particular, self-organizing, adaptive systems are designed to reconfigure their internal structure and improve their operating state at runtime. On the other hand, constraint programming and mathematical programming are successful paradigms designed to model a variety of satisfaction and optimization problems and solve them with generic algorithms.
Integration of both worlds promises interesting application and research areas with one major driving factor being the necessity to transform optimization models automatically due to the dynamic restructuring. We give an overview of existing techniques to use constraint programming in collective systems solving resource allocation problems and consider future extensions.

1 Introduction

Self-organizing, adaptive systems (SOAS) such as those considered in *organic computing* [24] benefit from redundancies introduced to assert a certain fault tolerance and the ability to stay functional in case of failures. Resource allocation problems such as the distribution of a given predicted energy demand to a set of power plants are instances of tasks that can benefit tremendously from a self-organizing structure to deal with uncertainty [2] and scalability [22]. Typically, SOAS are composed of a large number of autonomous entities that are capable of acting autonomously and self-organize. We therefore adopt the notion of "agents" to refer to these system components. However, in order to control these systems (or even understand their behaviour), we need to devise algorithms and techniques dealing with their complex nature.

Constraint programming (CP) is a flexible paradigm for solving a plethora of satisfaction and optimization problems [19]. A given problem is specified declaratively in terms of variables along with their domains and constraints that restrict valid assignments. General purpose solvers using systematic search (e.g., backtracking or branch-and-bound search algorithms) or local search techniques

enhanced by exact reasoning such as constraint propagation are then capable of solving these problems formulated in the abstract CP-formalism. Mathematical optimization, in particular linear and mixed integer programming, followings a similar paradigm for continuous domains such as, e.g., real-valued intervals.

These qualities, in particular the declarative design and the inherent generality, have been recognized to be useful for the specification and implementation of self-organizing, adaptive systems in Organic Computing in [17]: The authors introduced the *Restore-Invariant* approach to provide for rigorous specification of self-organizing systems. If a system leaves the so-called corridor of correct behaviour (i.e., by violating at least one of the constraints whose conjunction constitutes the *invariant*) the constraints are used to restore a functioning system state by solving a constraint satisfaction problem (CSP).

In this work, we emphasize a particular problem often solved by collective systems, *viz* resource allocation (see, e.g., [1, 9]) which appears naturally in the context of energy management systems as we illustrate in Section 2. The general one-good resource allocation problem without externalities [27] is stated as follows: Given a total quantity x_R of a resource, find an allocation $\langle x_1, \ldots, x_n \rangle$ of the resource to n agents to solve

$$\underset{\langle x_1,\ldots,x_n\rangle}{\text{minimize}} \quad \sum_{i=1}^{n} c_i(x_i) \quad \text{subject to} \quad \sum_{i=1}^{n} x_i = x_R \tag{1}$$

where $c_i(x_i)$ is a cost function for allocating x_i of the resource to agent i. Driven by requirements from our case study, we extend this definition to allocate not a single resource but a vector representing the resource to be distributed over the course of several time steps. The allocation problem then corresponds to finding vectors of contributions for each agent where consecutive contributions are limited due to physical constraints such as, e.g., a limited rate of change as Figure 3 shows.

To deal with the increasing complexity due to a rising number of included agents, a hierarchical decomposition strategy amenable to self-organization can be put in place [25], as illustrated in Figure 2. The model of one *intermediary* (i.e., an inner node in the tree-shaped hierarchy) then consists of a set of (a priori unknown) models representing its subordinate agents. The overall resource allocation problem presents itself at every inner node and can be solved recursively.

Figure 1 illustrates the overall "big picture" of the PhD-project including the fields and tasks that are encountered throughout the path to using constraint techniques in SOAS solving resource allocation problems in a hierarchical manner. This work emphasizes aspects of the "modeling" part and gives an outlook on future work in particular in the field of abstraction. Our roadmap leads from a modeling methodology to incorporate heterogeneous agents into one optimization problem over the specification of individual preferences to a automatically abstracting a less complicated model for decision making – motivated by the hierarchical decomposition strategy. Furthermore, we consider approaches to create

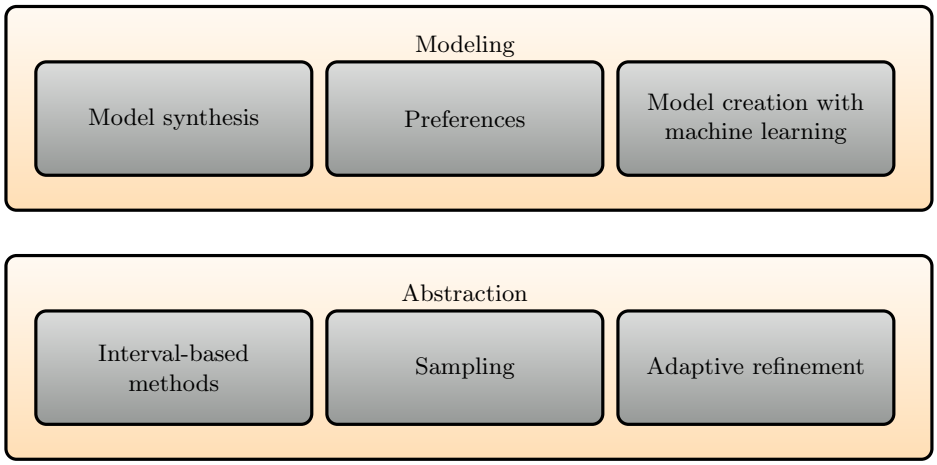


Fig. 1. A structural overview of the techniques involved to solve hierarchical resource allocation problems using constraints.

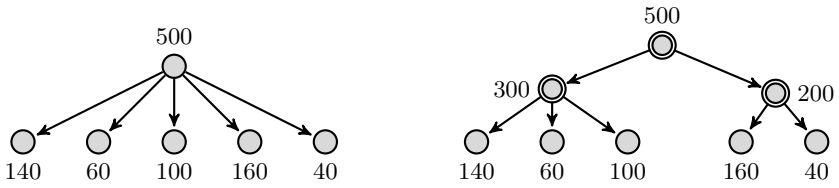


Fig. 2. Resource allocation problems can be solved using a hierarchical decomposition structure. Inner nodes representing intermediaries are marked by double circles.

constraint models from a set of observed data using machine learning as well as touching on techniques for preference elicitation.

2 Case Study: Power Plant Scheduling

We first illustrate our considerations with a simplified example taken from scheduling power plants in energy management [2, 22, 23] that corresponds to a dynamic hierarchical resource allocation problem. The resource to be distributed is given by the energy demand collected for a future timeframe that has to be allocated to a set of hierarchically organized power plants. It is dynamic in the sense that the problem has to be solved repeatedly over the course of time since predictions of demand and weather conditions affecting intermittent power generators such as solar plants change.

Power plants are modeled as agents that autonomously group to organizations, forming *Autonomous Virtual Power Plants* (AVPP) to participate, e.g., in a deregulated energy market [4]. Power plants face a set of physical restrictions

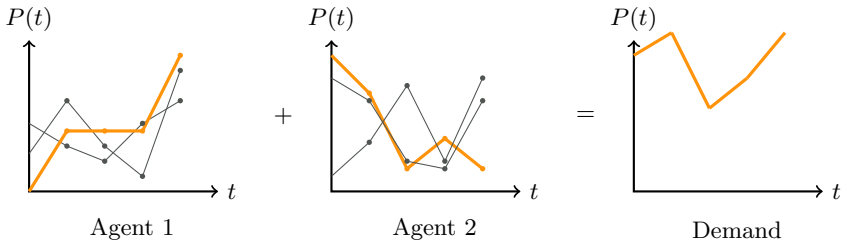


Fig. 3. Resource allocation over multiple time steps can be viewed as selecting the optimal combination of trajectories of two agents to meet a given demand.

regarding the possible trajectories of their output. These include minimal and maximal boundaries as well as limited rates of change due to inertia. This ramping behaviour needs to be taken into consideration in order to make efficient use of larger thermal units [5]. This fact enforces that some proactive scheduling has to be performed (since plants cannot arbitrarily change their contribution), resulting in a time-dependent resource allocation problem. Figure 3 illustrates how trajectories are selected from the respective sets to meet a given demand. To efficiently represent the set of possible trajectories, we use *hard* constraints to restrict the possible change between two outputs as well as general boundaries and frame the following overall optimization problem extending (1) (see [2] for details):

$$
\begin{aligned}
\underset{\mathbf{S}}{\text{minimize}} \quad & \alpha_{\Delta} \cdot \Delta + \alpha_{\Gamma} \cdot \Gamma \\
\text{subject to} \quad & \forall a \in \mathcal{A},\, t \in \mathcal{W} : \exists [x, y] \in L_t^a : x \leq S_t^a \leq y, \\
& v_{\min}^a(S_{t-1}^a) \leq S_t^a \leq v_{\max}^a(S_{t-1}^a) \\
\text{with} \quad & \Delta = \sum_{t \in \mathcal{W}} |\mathcal{R}_t - \sum_{a \in \mathcal{A}} S_t^a| \\
& \Gamma = \sum_{t \in \mathcal{W}} c_a\,(S_t^a)
\end{aligned}
\tag{2}
$$

where $\mathbf{S}$ is a vector representing the scheduled production and S_t^a stands for the production to be provided by agent a at time step t, taken from the respective sets $\mathcal{A}$ and $\mathcal{W}$ (the so-called scheduling *window*). Feasible regions for contributions are given by sets of intervals L_t^a and inertia is represented by the functions $v_{\min}, v_{\max}$ that map a scheduled output to the minimal (resp. maximal) output in the following time step. The objectives of the problem are then twofold: The combined production should meet the predicted energy demand $\mathcal{R}$ as closely as possible at all time steps (objective Δ) but also do this in an economical fashion (objective Γ, based on individual cost functions c_a). The relative importance can be tuned with the parameters α_{Δ} and α_{Γ}.

A subset of the possible trajectories satisfies also *soft* constraints such as preferred ranges of operation or limited (yet technically feasible) rates of change.

A system's designer may then relate soft constraints *qualitatively* to alleviate the need of specifying numerical weights. These qualitative preferences over soft constraints are specified as constraint relationships: e.g., "produce more than 500 MW" is more important than "change the output not faster than 5 MW in 15 minutes". Both constraints may though be dropped, if necessary.

In practice, however, the optimization problem is subject to several uncertainty factors such as varying actual productions, accuracy of predictions and the like. These are dealt with using trust models [2, 3] outside the scope of this work. Moreover, issues from the heterogeneity of the individual models and scalability concerns arise.

3 Synthesis of Heterogeneous Constraint Models

Obtaining concrete, solvable optimization problems based on the mathematical formulation in (2) gets complicated by the *heterogeneity* of agents. These different individual models need to be synthesized in order to acquire an optimization problem suitable for the collective – e.g., an organization representing its members.

The energy domain provides specific examples for heterogeneity since there exist power plants that can be switched on almost instantaneously in contrast to thermal power plants requiring a start-up phase that depends on the current down time. Table 1 shows exemplary start-up times for different types of power plants. Also, minimal up and down times can exist [5].

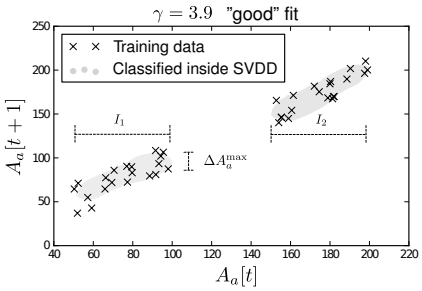
Some decision variables and constants may be required for *all* participants to represent the collective's goal (e.g., the maximal possible contribution of each agent in a resource allocation problem), whereas others only appear in some models and should thus be seen as "private" variables to be used for internal purposes – similar to the object-oriented approach in imperative programming.

To incorporate this heterogeneity in power plant models, additional decision variables to represent the down times in a schedule have to be introduced. With them, a transition system as given in Figure 4b can be specified in terms of corresponding constraints. We therefore distinguish between *interface* and *local* variables [22]. Future research could target a constraint language that is well-equipped for these practical necessities.

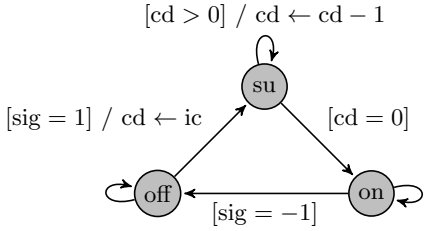
Figure 4b shows the transition system of a power plant during its start-up phase. The signal (sig) takes values from $\{-1, 0, 1\}$ where 1 shows a start-up, 0

Table 1. Different cold and hot start-up times in h for power plant types [13, 15]

Plant type	Cold start-up (down > 48h)	Hot start-up (down < 8h)
Black coal	4 – 5	2
Brown coal	6 – 8	2 – 4
Gas turbine	0.5	0.25
Photothermal	4 – 5	2



(a) A Constraint Model extracted from an SVDD trained on proposed data



(b) Transition system to model adaptive start-up times depending on down time.

Fig. 4. Learning models from auction proposals and representing power plants as transition systems.

is the default, and -1 triggers a shut-down. A countdown is initialised with *ic* (*initial counter*) as a result of a function of the down time and has the plant stay in the state *su* (start-up) for *ic* steps. A plant can only contribute in the state *on*. Once the power plant is in the state *on*, it can contribute energy in a certain range where maximal changes can be represented by additional, continuous, transitions. Interesting opportunities appear to emerge from modeling resource producers as transition systems including model checking or automata-based abstraction on composed systems and open up better testing and verification possibilities. Furthermore, it is possible that the model based view provides for an implementation of specific constraint propagators that can help improve the solving process.

4 Constraints and Preferences

In collective systems including a growing number of autonomously acting agents, it of paramount importance to have formalisms for dealing with *preferences* representing the agents' goals that can easily be combined. Moreover, realistic models can easily lead to over-constrained problems if mere preferences are treated as hard constraints. A formalism designed to deal with preferences in constraint problems *qualitatively* was first introduced in [21]: the so-called *constraint relationships*. Users specify a directed acyclic graph (dag) to denote which constraints are most important to be satisfied instead of using numerical weights. Section 5 discusses an idea for preference elicitation using this formalism.

We briefly revisit relevant definitions of constraint programming and constraint relationships that will serve for our discussions. A classical CSP is given by (X, D, C), where X denotes the decision variables to be assigned values from their domain D such that all constraints C are satisfied. An assignment $v \in [X \to D \cup \{?\}]$ is then a mapping that assigns each variable a value from the domain, where ? stands for "unassigned". Hard constraints then map assignments to $\mathbb{B}$, denoting whether the constraint holds. Consider, e.g., the con-

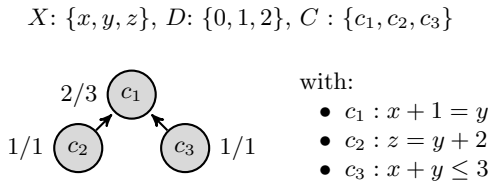


Fig. 5. Not all three constraints can be satisfied simultaneously, e.g. c_2 forces z to be 2 and y to be 0, conflicting with c_1, We can choose between solutions satisfying $\{c_1, c_3\}$ or $\{c_2, c_3\}$. Weights are given for SPD and TPD.

straint $c_1 : x < y$: An assignment $v_1 = \{x \mapsto 1, y \mapsto 2\}$ satisfies c_1, whereas $v_2 = \{x \mapsto 1, y \mapsto 1\}$ does not. In practice, however, this definition may be too rigid as problems can be over-constrained, i.e., no assignment exists that satisfies *all* constraints (a *solution*). To cope with these situations, *soft constraints* have been introduced [16] with a generalizing framework given by c-semiring based formalisms [6]. Constraint relationships can be considered a new instance of this soft constraint framework [14].

4.1 Constraint Relationships

Constraint relationships provide a *qualitative* alternative to soft constraints – as opposed to weighted CSP, fuzzy CSP, or constraint hierarchies. They are similar in nature to CP-nets [7] but formally incomparable to them [21]. Users specify their preferences over constraints to be satisfied in form of a dag $(C, \rightarrow_C)$, where C (the set of constraints) is the carrier of the dag, i.e., the vertices; an edge $c \rightarrow_C c'$ indicates that c "precedes" c' and is thus deemed *less* important. Figure 5 shows a toy example of a CSP with constraint relationships.

An interesting question is how to lift the order induced over constraints by the dag to a *set* of violated constraints representing an assignment. Put differently, we need to specify *how* much more important a certain constraint is with respect to a set of other constraints. A rather obvious lifting respects set inclusion, i.e., violating a superset of constraints should always worsen the quality of an assignment. In addition to that, we proposed different dominance properties in [21], where SPD (single-predecessor-dominance) indicates that one constraint may only dominate a single predecessor and TPD (transitive-predecessor-dominance) defines that a single constraint is more important than *all* of its predecessors (which is identical to the semantics used in constraint hierarchies). It turns out that the single-predecessor lifting corresponds to the ordering obtained by an algebraically *free* construction from a partial order to a c-semiring [14]. Research on this lifting motivated a more general representation of soft constraints based on partially ordered monoids that was independently found by [11] for lexicographic combinations. In [21], we also gave a mapping from constraint relationships to weighted CSP that can be directly used with standard constraint solvers. Future work may refresh discussions about soft constraint frameworks in terms of

common solver development and particularly emphasize the need of preference management in multi-agent systems to model, e.g., strategic behaviour [20] using voting and game theory.

4.2 Preference Elicitation using Abductive Reasoning

Since constraint relationships hope to provide a conceptually simpler formalism towards users, the preference elicitation process could be supported efficiently. Typically, users are shown solutions and may decide which they prefer. A preference model is then sought after. In particular, it is paramount to be able to reconstruct a preference decision, based on the rules for constraint relationships.

Assuming that preferences can be expressed easier when looking at *solutions*, we plan to offer a tool based on *abductive reasoning* [10](i.e., given a proof goal in logical programming – add the necessary facts to a database) solving an inverse problem to standard constraint relationships: Users specify their preference over sets of violated constraints and the system tries to find a consistent constraint relationship that explains the decisions made by the user. A prototypical output from this system implemented in Prolog+CHR illustrates our idea:

```
?- worsens([a,b], [b,c]), worsens([b], [a]).
crbetter(c,b)
crbetter(a,b)
crbetter(c,a)
```

If presented with the facts that a user prefers a set of violated constraints $\{b, c\}$ to $\{a, b\}$ and $\{a\}$ to $\{b\}$, it correctly finds a matching constraint relationship. We envision an interactive and incremental preference elicitation process to be used in conjunction with preferences of consumers or producers of energy.

5 Model Creation by means of Machine Learning

Our approach so far assumed that formal models of an agent's behavior are available for all contributing agents. It is easy to envision situations where no neatly specified constraint models exist, be it for privacy motivated (e.g., a private household offering their biogas power plant does not want to disclose how low they can be run) or practical reasons (e.g., no formally explicit but only a simulation model of the system exists and validity is judged by several simulation runs [8]). In these case, we want to provide techniques that can at least approximately learn a sufficient control model.

A first approach is currently in progress in combination with a market-based algorithm for the resource allocation problem faced in energy management [4]. Pairs $\langle A_a[t], A_a[t+1]\rangle$ represent consecutive *actual* contributions of agent a that were observed and now constitute the training set. Figure 4a illustrates how control constraints are to be extracted from a support vector domain description classifier [26]. We can read off feasible intervals as well as limited rates of change due to inertia. Future work should examine alternative, perhaps probabilistic, machine learning techniques to extract different model elements.

6 Model Abstraction

As mentioned before in Figure 2, once the complexity of the synthesized optimization problem increases beyond being efficiently solvable, we employ a hierarchical decomposition strategy [25]. To reduce the complexity, we need to calculate *abstracted* models that represent the possible transitions of an intermediary – based on its subordinate agents. Certainly, this problem is combinatorial in nature as, e.g., a contribution of 500 might be achieved by several internal configurations (one agent contributing 200, the other 300, etc.).

In [2] and [22], we described an initial set of constraint model abstraction techniques. Future work needs to attempt applying abstraction to more complex agent models and other problems where new abstraction algorithms need to be devised (such as the abstraction of consumer behaviour). The general idea is that model abstraction is performed bottom-up before the resource allocation itself may be done top-down in a recursive manner.

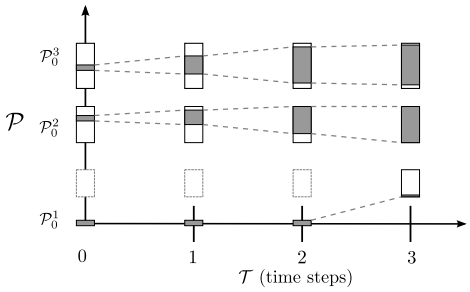
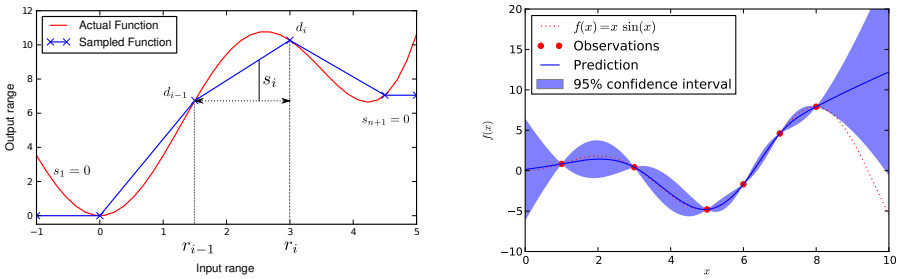


Fig. 6. Temporal abstraction finds boundaries for the contributions of contributions based on the current state of its subordinate agents.

To illustrate these concepts, we briefly revisit three consecutive steps, based on a set of (subordinate) agent models:

1. *General Abstraction:* First, we determine *generally* feasible contribution ranges. This is done by means of interval arithmetics using combination and merge operations to abstract from the internal exact configuration. For instance, when combining two agents p and q capable of providing $L^p = \{[0,0],[1,4]\}$ and $L^q = \{[0,0],[7,10]\}$, then the intermediary a can contribute in any range appearing in the Cartesian product of both contribution interval sets, i.e., $L^a = \{[0,0],[1,4],[7,14]\}$.
2. *Temporal Abstraction:* Afterwards, the so-called *temporal abstraction* restricts feasible contributions on the basis of the current *state* and constraints on the possible behavior of the subordinate up to a specified point in time. This is done by minimizing and maximizing the contribution based on the current step and then performing similar interval calculation as in the general case. Figure 6 illustrates that algorithm.



(a) A piecewise linear function approximating some collective's properties.

(b) Gaussian Process regression models offer uncertainty about the value $f(x)$ of a learned function.

Fig. 7. Using sampling abstraction and piecewise linear functions.

3. *Sampling Abstraction:* To obtain simplified representations of functional relationships between decision variables, we use a sampling based approach where we have to solve slightly modified optimization problems repeatedly. For instance, to obtain a mapping from production to costs in an AVPP, we would solve the problems "minimize cost if the production is x_i", with x_i taking values from the minimal to the maximal output (obtained by general abstraction). We collect the results r_i along with x_i to store (x_i, r_i) pairs that are used for a piecewise linear function, as Figure 7a shows.

Since first results where promising [22], we want to support more complex (e.g., automaton based) models and improve the quality of abstractions. The quality of sampled functions typically improves by selecting additional supporting sampling points. But each sampling point corresponds to an optimization run such that the costs of abstraction might invalidate the benefits. We therefore envision two possible enhancements of the algorithms:

1. We want to select the most useful or informative points. This corresponds to the task faced in active learning, where input values are selected to be labeled. We want to investigate how algorithms exploiting adaptive submodularity [12] based on Gaussian processes (see 7b) improve the selection of sampling points.
2. Over the course of several runs, abstraction errors might appear in certain contribution ranges or particular transitions more frequently (e.g., due to the underlying cost functions) whereas others are not involved. Therefore, the abstraction can adaptively refine over the course of time, similar to ideas used in CEGAR loops with hybrid systems [18].

7 Conclusion

We have shown several techniques under the umbrella of constraint programming for hierarchical resource allocation. The overall problem and the included

techniques constitute the backbone of the PhD-project. This paper emphasized several ideas relating to new aspects of modeling agents to be included in a resource allocation problem, taking into account individual technical aspects and preferences. Furthermore, we presented first ideas to create system models from observations using machine learning in case no explicit control models exist.

Future work includes improving the abstraction algorithms to be more efficient and accurate as well as applying the algorithms and models to problems outside the power management case study to inspect the generality.

On the one hand, specific problems occurring in this domain can be tackled using CP, on the other hand, new languages and formalisms can improve the CP-framework itself and stipulate interesting research fields. We hopefully motivated that there is potential for a fruitful integration of these fields.

References

1. Abouelela, M., El-Darieby, M.: Multidomain hierarchical resource allocation for grid applications. Journal of Electrical and Computer Engineering 2012 (Jan 2012)
2. Anders, G., Schiendorfer, A., Steghöfer, J.P., Reif, W.: Robust Scheduling in a Self-Organizing Hierarchy of Autonomous Virtual Power Plants. In: Proc. of the 2nd International Workshop on Self-optimisation in Organic and Autonomic Computing Systems (SAOS14) in conjunction with ARCS 2014. vol. 2 (February 2014)
3. Anders, G., Siefert, F., Steghöfer, J.P., Reif, W.: Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-Organizing Systems. In: Elmenreich, W., Dressler, F., Loreto, V. (eds.) Self-Organizing Systems – Proc. of the 7th International Workshop on Self-Organizing Systems (IWSOS 2013). Lecture Notes in Computer Science, vol. 8221, pp. 90–102 (May 2013)
4. Anders, G., Steghöfer, J.P., Siefert, F., Reif, W.: A Trust- and Cooperation-Based Solution of a Dynamic Resource Allocation Problem. In: Proc. 7$^{\text{th}}$ IEEE Int. Conf. Self-Adaptive and Self-Organizing Systems (SASO'13). IEEE (2013)
5. Arroyo, J., Conejo, A.: Modeling of start-up and shut-down power trajectories of thermal units. Power Systems, IEEE Transactions on 19(3), 1562–1568 (2004)
6. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimization. J. ACM 44(2), 201–236 (1997)
7. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. J. Artif. Intell. Res. 21, 135–191 (2004)
8. Bremer, J., Rapp, B., Sonnenschein, M.: Encoding distributed search spaces for virtual power plants. In: Computational Intelligence Applications In Smart Grid (CIASG), 2011 IEEE Symposium on. pp. 1–8 (April 2011)
9. Chevaleyre, Y., Dunne, P.E., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J.A., Sousa, P.: Issues in multiagent resource allocation. Informatica 30(1), 3 – 31 (2006)
10. Christiansen, H., Dahl, V.: Hyprolog: A new logic programming language with assumptions and abduction. In: Logic Programming, pp. 159–173. Springer (2005)
11. Gadducci, F., Hölzl, M.M., Monreale, G.V., Wirsing, M.: Soft Constraints for Lexicographic Orders. In: Castro, F., Gelbukh, A., Gonzlez, M. (eds.) Proc. 12$^{\text{th}}$ Mexican Int. Conf. Artificial Intelligence (MICAI'13). pp. 68–79. Lect Notes Comp. Sci. 8265, Springer (2013), http://dx.doi.org/10.1007/978-3-642-45114-0_6

12. Golovin, D., Krause, A.: Adaptive submodularity: Theory and applications in active learning and stochastic optimization. J. Artif. Int. Res. 42(1), 427–486 (2011)
13. Jarass, L., Obermair, G.: Welchen Netzumbau erfordert die Energiewende?: Unter Berücksichtigung des Netzentwicklungsplans Strom 2012 (in German). MV-Wissenschaft, Monsenstein und Vannerdat (2012)
14. Knapp, A., Schiendorfer, A.: Embedding Constraint Relationships into C-Semirings. Tech. Rep. 2014-03, Institute for Software and Systems Engineering, University of Augsburg (2014)
15. Mayer, J.N., Kreifels, N., Burger, B.: Kohleverstromung zu Zeiten niedriger Börsenstrompreise (Aug 2013)
16. Meseguer, P., Rossi, F., Schiex, T.: Soft Constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 9. Elsevier (2006)
17. Nafz, F., Seebach, H., Steghöfer, J.P., Anders, G., Reif, W.: Constraining self-organisation through corridors of correct behaviour: The restore invariant approach. In: Mller-Schloer, C., Schmeck, H., Ungerer, T. (eds.) Organic Computing A Paradigm Shift for Complex Systems, Autonomic Systems, vol. 1, pp. 79–93. Springer Basel (2011), http://dx.doi.org/10.1007/978-3-0348-0130-0_5
18. Prabhakar, P., Duggirala, P., Mitra, S., Viswanathan, M.: Hybrid automata-based cegar for rectangular hybrid systems. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, Lecture Notes in Computer Science, vol. 7737, pp. 48–67. Springer Berlin Heidelberg (2013)
19. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Foundations of Artificial Intelligence, Elsevier Science (2006), http://books.google.de/books?id=Kjap9ZWcKOoC
20. Rossi, F.: Collective decision making: a great opportunity for constraint reasoning. Constraints 19(2), 186–194 (2014), http://dx.doi.org/10.1007/s10601-013-9153-3
21. Schiendorfer, A., Steghöfer, J.P., Knapp, A., Nafz, F., Reif, W.: Constraint relationships for soft constraints. In: Bramer, M., Petridis, M. (eds.) Research and Development in Intelligent Systems XXX. Springer London (2013)
22. Schiendorfer, A., Steghöfer, J.P., Reif, W.: Synthesis and Abstraction of Constraint Models for Hierarchical Resource Allocation Problems. In: Proc. of the 6th International Conference on Agents and Artificial Intelligence (ICAART). vol. 2. SciTePress (March 2014)
23. Schiendorfer, A., Steghöfer, J.P., Reif, W.: Synthesised constraint models for distributed energy management. In: Proc. of the 3rd International Workshop on Smart Energy Networks & Multi-Agent Systems (SEN-MAS) in conjunction with FedCSIS 2014 (Sept 2014)
24. Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., Richter, U.: Adaptivity and self-organization in organic computing systems. ACM Trans. Auton. Adapt. Syst. 5(3), 10:1–10:32 (Sep 2010), http://doi.acm.org.eaccess.ub.tum.de/10.1145/1837909.1837911
25. Steghöfer, J.P., Behrmann, P., Anders, G., Siefert, F., Reif, W.: HiSPADA: Self-organising hierarchies for large-scale multi-agent systems. In: Proceedings of the IARIA International Conference on Autonomic and Autonomous Systems (ICAS) 2013. IARIA (2013)
26. Tax, D.M., Duin, R.P.: Support vector data description. Machine learning 54(1), 45–66 (2004)
27. Van Zandt, T.: Hierarchical computation of the resource allocation problem. European Economic Review 39(3-4), 700–708 (April 1995)