On Automated Design in Chemical Engineering [*]

André Schulz Benno Stein [†] Annett Kurzok [‡]

Abstract

Design support in chemical engineering is typically connected with the configuration of the technical devices that realize a chemical process. In fact, the configuration step is a rather finishing job within the entire design process, and several hurdles have been taken by the designer before.

This paper outlines both an idea and a methodology to realize a design support right from the start of the design process. Central element of the methodology is the abstraction of a chemical process as a graph along with the use of graph grammars, which encode an engineer's design knowledge. By applying graph transformation rules, an incompletely and coarsely defined design can be completed and refined towards a desired solution.

1 Introduction

In order to facilitate the work in the field of chemical engineering, computers have been repeatedly successfully deployed. The computer support encloses not only the configuration of particular devices such as mixers [2, 5] by means of expert systems, but also the use of substance databases, simulation tools, or tailored CAD programs [11, 6, 12, 9, 8].

In contrast to these approaches, the paper in hand aims at a holistic support of the design procedure at the level of parameterized unit-operations. Typical unit-operations are mixing (homogenization, emulsification, suspension etc.), heat transfer and flow transport (pumps). Given a description of the substance inputs and the substance outputs, both the selection and the arrangement of the necessary unit-operations shall be derived.

A support at this level is interesting within the following respects:

1. Structure and type of unit-operations depend on each other. A holistic view provides means to detect and to process interdependencies between parts of a chemical process.
2. Having no structure predefined, a chemical process can be synthesized from scratch. And, at the abstraction level of unit operations, such a synthesis becomes tractable.
3. Our approach can provide insights and methodologies necessary for an entire automation of the chemical design process, which may become subject to future research.
4. Existing approaches to design support concentrate on isolated design aspects only.

Key concepts of our approach are: A chemical process is viewed as a graph. The nodes of the graph describe unit-operations, the edges of the graph specify both the flow and the properties of the processed substance. Modifications of a chemical process can then be defined as node-insertion and node-deletion operations on the "chemical graph". Since we are working on a graph, graph grammars provide a proper means to precisely specify such modifications, say, to encode an engineer's design knowledge.

The paper is organized as follows. Section 2 gives a description of the chemical design task both from a theoretical side and exemplified at a realistic example. Section 3 then introduces the concepts of a graph and a graph grammar tailored to our design problem. Section 4 picks up the introductory example and shows how it is solved by our graph grammar approach. The last section, 5, elaborates on some theoretical aspects of our approach.

The presented approach may be suited to tackle various kinds of chemical design problems. However, within our project as well as our research we have restricted ourselves to a particular part of chemical processes: The design of plants for food processing.

2 The Design Task

The task of designing a chemical plant is defined by the given input and the desired output. The goal is to mix or transform various input substances in such a way that the resulting product meets the imposed requirements. In general, this process may also yield some by-products, but this will be neglected in this paper—we will restrict ourselves to the $n : 1$ case. Figure 1 illustrates the solution process followed in general practice.

[*] Supported by DFG grant PA 276/23-1

[†] Department of Computer Science, University of Paderborn, 33095 Paderborn, Germany, email: {aschulz,stein}@upb.de

[‡] Department of Engineering, University of Paderborn, 33095 Paderborn, Germany, email: annett.kurzok@vt.upb.de

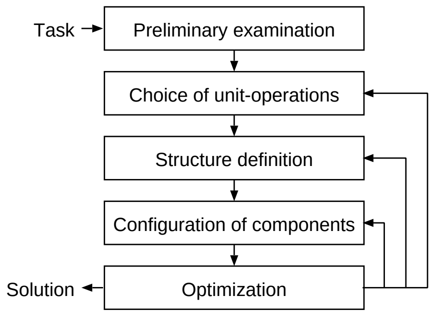


Figure 1: Steps in the design process.

The steps depicted on figure 1 can be described in more detail as follows:

1. *Preliminary examination*. The task specification, i. e., the input substances, the desired output and any additional requirements, is analyzed.

2. *Choice of unit-operations*. Having examined the substances involved in the process, abstract building blocks, so-called unit-operations (or unit-ops), are chosen in compliance with certain rules.

   In practice, engineers choose concrete devices at this stage—the use of abstract building blocks is done implicitly.

3. *Structure definition*. The previous step yields a set of unit-ops devoid of any structure. In order to find an apt topology, different circuits are tried until one that meets the requirements is found. This well-known *propose-and-revise* behavior has been also applied by [2].

4. *Configuration of devices*. The chosen devices, still represented by unit-ops, are instantiated. Beginning with the first unit-op in the process, concrete devices are chosen from a database. Since different devices of the same class often produce outputs with deviating properties, these changes must be propagated, thus influencing the choice of later devices.

5. *Optimization*. The plant's functionality is tested whether it meets the requirements. If the design fails to fulfill any of these conditions, changes have to be applied either to the structure or to the set of chosen devices.

   Even if the plant represents a solution, the engineer still refines it to reduce energy consumption or to decrease mixing time. After a modification a jump back to a previous step may become necessary.

2.1 Example

The following task specification excerpt shall illustrate the usual design procedure.

Name	Mass	Temp.	Viscosity
sugar	47.62%	20 C	–
water	15.75%	20 C	0.0010012 Pas
starch syrup	36.63%	20 C	0.2-1.6 Pas
caramel syrup	100.00%	110 C	?

The goal is to produce caramel syrup, which is necessary for the production of caramel bonbons, using water, starch syrup, and sugar.

The following table of viscosity values of sugar solution, an intermediate product, is also given:

Temperature	Viscosity (71% solution)
20 C	500 Pas
30 C	250 Pas
40 C	130 Pas
50 C	80 Pas
60 C	50 Pas
70 C	30 Pas
80 C	20 Pas

Based on this specification, the following is done in compliance with the steps depicted above:

1. *Preliminary examination*. Sugar is a solid and must be dissolved within a liquid. Water has a lower viscosity than starch syrup, so it is better to mix sugar and water first and then add the starch syrup. Depending on the mass ratios the water may have to be heated up to increase solubility.

2. *Choice of unit-operations*. Comparing the mass ratios of sugar and water leads to the conclusion that heating is needed; therefore, a heat transfer unit-op is added. To mix the heated water and sugar, a mixing unit-op for lower viscose substances is fitting. To avoid recrystallization, the starch syrup should also be heated, making a further heat transfer unit-op necessary. Lastly, the sugar solution and the heated starch syrup are mixed. In order to reach 110 C, another heat transfer unit-op is needed. Besides, pump unit-ops are required to transport the substances between devices.

3. *Structure definition*. The choice of unit-ops allows for conclusions pertaining to the arrangement of the unit-ops. In our example the arrangement is relatively evident; figure 2 shows the chosen topology.

4. *Configuration of devices*. Based on the mass, the volume, and the other properties of the involved substances, matching devices are chosen from databases.

5. *Optimization*. The computed parameters of the design are usually feasible values, but improvement can still be achieved. With this goal in mind, the parameterization step is repeated and parameters are adjusted. By limiting the last mixing unit-op to devices with a built-in heat transfer unit, the last heat transfer unit-op becomes superfluous. This change shortens the process chain (see figure 3).

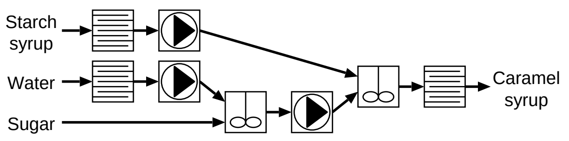


Figure 2: The first design of the example process.

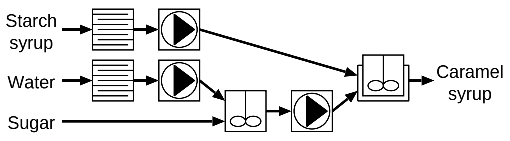


Figure 3: The final design of the example process.

Alternatively, a design with fewer devices is possible: Water and starch syrup could be mixed first, and the resulting solution used to dissolve sugar. This structure choice requires one heat transfer unit-op less than the proposed design because both water and the starch syrup can be heated together. However, this alternative would cause a longer mixing time, since the sugar must be dissolved in a more viscose solution.

3 Design with Graph Grammars

The steps listed in section 2 can be automated in an isolated fashion. However, a separate processing may lead to loss of information, since the choice of a unit-op often affects the structure and vice versa. For example, the choice of a certain mixer might influence the decision whether a heat transfer device is needed, possibly changing the topology. Likewise, a certain order of devices can make one of them superfluous.

Since these steps are intertwined, it is desirable to combine the choice of unit-ops and the structure definition to make use of all information available. One way of tackling both tasks simultaneously is to use a graph grammar to generate feasible designs, which are generated in a controlled manner. The following subsections introduce the approach.

3.1 Graph Transformation

Figures 4 and 5 show two examples of rules that may be used to transform an abstract design (left-hand side) towards a refined design (right-hand side).

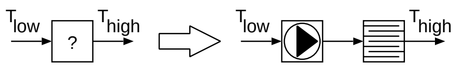


Figure 4: Replacement of an unknown unit by inserting heat transfer and pump units to fulfill the temperature constraints.

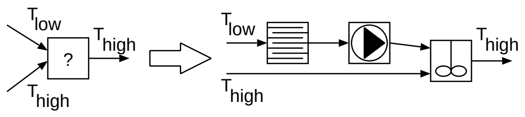


Figure 5: Replacement of an unknown unit by inserting a heat transfer unit, a pump unit and a mixing unit. Note that the label, number and directions of edges is obeyed..

What happens, from a graph-theoretical point of view, is that a node t in the original graph G is replaced by a graph R. Put another way, R is embedded into G. Figure 6 depicts two samples for two different graphs R.

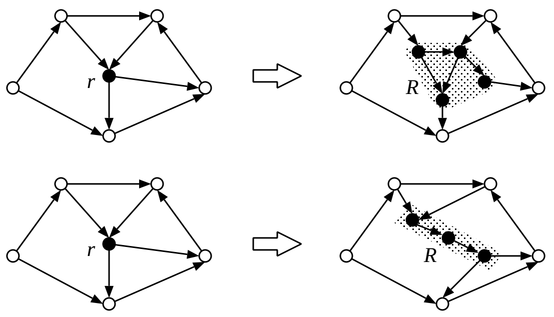


Figure 6: Replacement of node t by a new graph R..

In the following we will shortly outline a formal basis for the illustrated graph transformation.

Definition 1 *A labeled graph is a tuple $G = \langle V_G, E_G, \sigma_G, \gamma_G \rangle$ where*

- *V_G is the set of nodes,*
- *$E_G \subseteq V_G \times V_G$ is the set of directed edges,*
- *σ_G is the node label function, $\sigma_G : V_G \to \Lambda$, where Λ is a set of symbols, called the label alphabet,*
- *and γ_G is the edge label function, $\gamma_G : E_G \to \Lambda$.*

Notation: (v_1, v_2) represents a directed edge with tail v_1 and head v_2.

Informally, a graph grammar is a system that, applied to a graph G, yields a new graph G'. The following definition reflects the essence of this concept. Refer to [10] for further details.

Definition 2 *A graph grammar is a tuple* $\mathcal{G} = \langle \Sigma, \Delta, \Gamma, \Omega, P, s\rangle$ *with*

- Σ *is the alphabet of node labels,* $\Delta \subset \Sigma$ *is the set of terminal node labels,* $\Sigma \setminus \Delta$ *is the set of non-terminal node labels,*
- Γ *is the alphabet of edge labels,* $\Omega \subset \Gamma$ *is the set of terminal edge labels,* $\Gamma \setminus \Omega$ *is the set of non-terminal edge labels,*
- P *is the finite set of productions,*
- *and* s *is the initial symbol.*

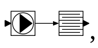
The productions of the set P *are tuples of the form* $t \to \langle R, I\rangle$ *with*

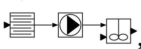
- $t \in \Sigma$ *is a label belonging to a node* $v \in V_G$*,*
- $R = \langle V_R, E_R, \sigma_R, \gamma_R\rangle$ *is the non-empty replacement graph. The nodes of* R *that are to be connected to the host graph are called cut nodes. To each cut node belongs at least one embedding specification* $i \in I$*.*
- I *is the embedding specification that consists of tuples* (l, s)*, where*
  - $l \in \Gamma$ *is an edge label,*
  - $s \in \Sigma$ *is the label of a cut node in the replacement graph* R*.*

  An embedding rule is interpreted as follows: Each edge labeled l *connecting a node* $v \in V_G$ *with the node labeled* t *is substituted by an edge labeled* l *connecting* v *to a cut node of* R *labeled with* s*.*

Example. In the following, we give a complete graph grammar. Among others, this graph grammar can be used to perform the design refinements shown in figures 4 and 5 respectively.

- $\Sigma \setminus \Delta = \{?\}$,
- $\Delta = \{$ mixer, pump, heat-transfer$\}$,
  $\Omega = \{T_{\text{low}}, T_{\text{high}}, \mathit{Visc}_{\text{low}}, \mathit{Visc}_{\text{high}}, \mathit{Density}_{\text{low}}, \mathit{Density}_{\text{high}}\}$
- $P = \{$*heater-rule, mixer-rule*$\}$, with

  heater-rule := (?, , $\{(T_{\text{low}}$, mixer), $(T_{\text{high}}$, heat-transfer)$\}$),

  mixer-rule := (?, , $\{(T_{\text{low}}$, heat-transfer), $(T_{\text{high}}$, mixer)$\}$)

4 Example

In section 2 we described the design procedure for a caramel syrup process and presented a solution to this problem. Now, we will use a graph grammar to attain the same goal. For this purpose, the graph rules depicted by figures 7 – 12 are given. Finally, figure 13 shows a derivation that produces a feasible design.

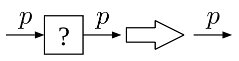


Figure 7: (**R1**) Deletion of non-terminal node.

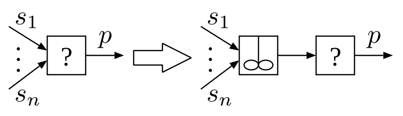


Figure 8: (**R2**) Insertion of a mixing unit-op.

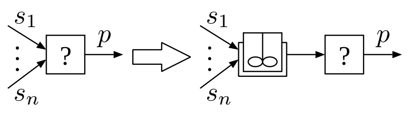


Figure 9: (**R3**) Insertion of mixing unit-op with built-in heat transfer unit.

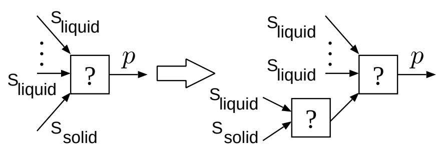


Figure 10: (**R4**) Improvement of mixing properties by handling solid inputs separately.

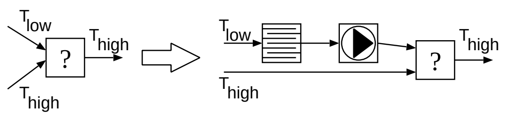


Figure 11: (**R5**) Improvement of dissolving properties by heating an input.

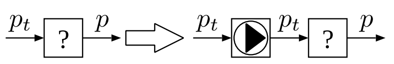


Figure 12: (**R6**) Insertion of a pump unit-op.

In general there will be a series of rules that apply for a given situation, leading to different solutions of varying quality and cost. Thus, the generation process can be viewed as a tree containing derivations for all possible alternatives, as shown in figure 14. Note that the graph grammar derivation of figure 13 corresponds to one branch of this tree.

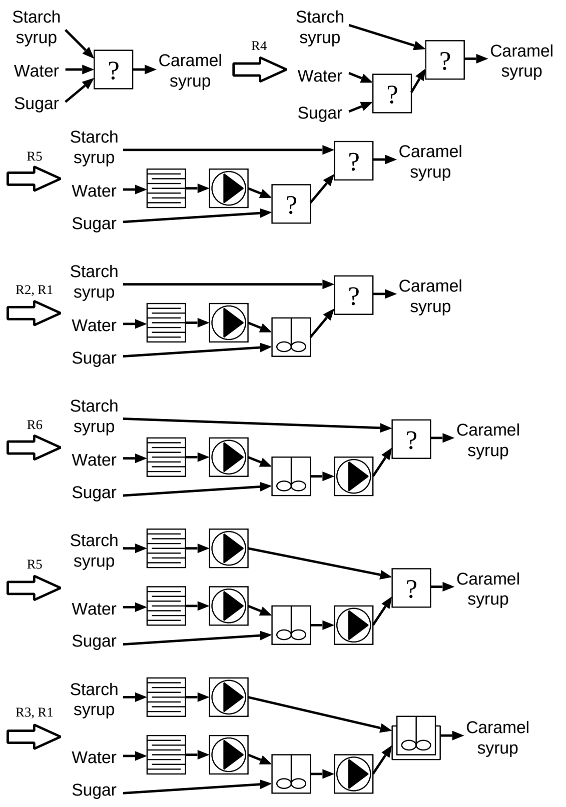


Figure 13: Derivation of the caramel plant design.

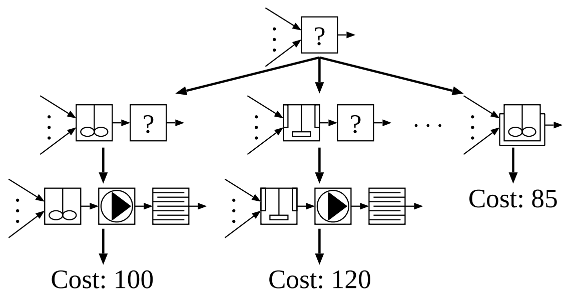


Figure 14: Search tree for the optimization of the design generation process.

5 Discussion

In this section, some issues that have direct consequences for the usability of the proposed approach described in section 4 are analyzed in detail.

5.1 Graph Topology

In practice, plants often combine various chemical processes within one single "chain" producing one main product and a series of by-products. This means that the topology represents at least a directed acyclic graph, which can only be generated by a context-dependent graph grammar[1]. However, such grammars imply exponential time complexity, as rules may have more than one non-terminal on the left-hand side (graph matching problem, see [7]).

[1]In the field of formal languages, this is known as *context-sensitivity* [4]

In order to avoid this drawback, we restrict—as far as possible—the set of graph production rules to context-free rules, which generate a graph in polynomial time [1].

5.2 Computational Complexity

The efficiency of the presented approach in terms of $\mathcal{O}$-notation [3], decides on its aptitude for practical use. In our case we are interested in the computational complexity in terms of the generated graph size, which is strongly related to the types of rules allowed.

- *Rule applicability*. The graph grammar concept, as presented in section 3, does not impose any restriction upon the rules as far as firing is concerned. In fact, the example of section 4 contains three different types of rules: rules that fire only once, e. g. *R1*, rules that fire linearly in the number of inputs, e. g. *R4*, and rules that can fire arbitrarily often, e. g. *R6*. The existence of the last rule type implies that the graph rule system may not terminate.

  However, the above drawback can be avoided by forbidding the repeated use of the same rule within the same context—in fact, graph grammar implementations do include facilities to specify forbidden and allowed rules (see *programmed graph replacement systems* in [10]).

- *Simple rules.* Only rules with bounded number of inputs and outputs are allowed; to be more precise, rules may only produce a single output, and rules may not use multiple inputs to produce further non-terminal nodes, such as in rule *R4*. Furthermore, we forbid cycles within the generated design, thus avoiding the repeated execution of rule sequences.

  Due to these restrictions, the size of designs generated by these rules are linearly related to the number of inputs available. Therefore, the computational effort to produce a feasible design is of the order $O(n)$.

- *Rules acting on multiple inputs.* Now we drop the restriction on rules to handle multiple inputs, i. e., rules such as *R4* are allowed. Depending on the number of inputs, further non-terminal nodes may be produced. This results in $O(n^2)$ computational effort.

- *Unrestricted rules.* Lastly, rules that multiply the number of outputs are also allowed. Since with these rules arbitrarily many new "inputs" can be generated, the computational effort is unbounded.

- *Configuration search strategy*. The generation of optimal designs requires an appropriate search strategy. Due to the nature of graph rule derivation, DFS (or BFS) with backtracking seem fitting strategies. To ensure better performance, further concepts such as *backjumping*, *branch-and-bound* and *iterative deepening* could be considered.

6 Summary

The paper introduced a new method to automate the design process of industrial food processing plants. In contrast to existing approaches, our method aims at a holistic support of the design procedure, at the level of parameterized unit-operations.

A chemical process is modeled as a graph whose nodes describe the unit-operations and whose edges specify the properties of the processed substance. Modifications of a chemical process are defined as node-insertion and node-deletion operations, which in turn are formalized by means of graph grammars.

In this way, design solutions for a chemical processing problem can be produced automatically by applying graph production rules that encode an engineer's design knowledge. This approach has a theoretically unbounded runtime behavior, but, if rules are formulated in compliance with the domain knowledge available, can be very efficient.

Anyway, our approach shall provide insights and evaluate methodologies necessary for an entire automation of the chemical design process, which may become subject to future research.

References

[1] F. J. Brandenburg. Designing graph drawings by layout graph grammars. In R. Tamassia and I. G. Tollis, editors, *Proc. DIMACS Int. Work. Graph Drawing, GD*, number 894 in Lecture Notes in Computer Science, LNCS, pages 416–427, Berlin, Germany, 1994. Springer-Verlag.

[2] A. Brinkop and N. Laudwein. Konfigurieren von industriellen rührwerken. *KI*, 2:54–59, 1993.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd. edition, 1994.

[4] J. E. Hopcroft. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[5] A. Knoch and M. Bottlinger. Expertensysteme in der verfahrenstechnik – konfiguration von rührapparaten. *Chem.-Ing.-Tech.*, 65(7):802–809, 1993.

[6] W. Marquardt. Rechnergestützte erstellung verfahrenstechnischer prozeßmodelle. *Chem.-Ing.-Tech.*, 64(1):25–40, 1992.

[7] K. Mehlhorn. *Data Structures and Algorithms*, volume 2 Graph Algorithms and NP-Completeness. Springer, Berlin, 1984.

[8] C. C. Pantelides. Speedup – recent advances in process simulation. *Comput. chem. Engng.*, 12(7):745–755, 1988.

[9] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. Ascend: An object-oriented computer environment for modeling and analysis: The modeling language. *Computers chem. Engng.*, 15(1):53–72, 1991.

[10] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1 Foundations. World Scientific, 1997.

[11] S. Räumschüssel, A. Gerstlauer, E. D. Gilles, B. Raichle, M. Zeitz, and W. Marquardt. An architecture of a knowledge-based process modeling and simulation tool. In *Proceedings IMACS/IFAC Second International Symposium on Mathematical and Intelligent Models in System Simulation*, volume 2, pages 242–247, 1993.

[12] G. Stephanopoulos, G. Henning, and H. Leone. Model.la. a modeling language for process engineering - i. the formal framework. *Computers chem. Engng.*, 14(1):813–846, 1990.