

# An integrated approach to bioprocess recipe design

Juho Rousu\* and Robert Aarts<sup>1</sup>  
 VTT Biotechnology, P.O. Box 1500, FIN-02044 VTT,  
 Finland

Biotechnical and food process industries have a need for tools that support the implementation of modern manufacturing practice. Flexible production strategies and total quality management require efficient design of the process recipes.

This paper presents the approach used in Sophist, an adaptive recipe planning system. The system integrates case-based reasoning and a semi-qualitative process model for the design and optimisation of bioprocess recipes. In particular, we introduce a novel revision selection algorithm that improve the system's adaptivity. The behavior of the planner is exemplified by a batch fermentation planning experiment.

## 1. Introduction

Biotechnical and food process industries have a need for tools that support the implementation of modern manufacturing practice. Flexible production strategies and total quality management require efficient design of the process recipes. This task entails selecting appropriate values for those process parameters that need to be planned before every batch of the process. Examples of such parameters are the choice and amounts of ingredients, set-point profiles, process start and finish times, etc. Current practice is to modify the recipes only when several, consecutive, batches yield unsatisfactory results.

At the same time, the food industry has an underutilized resource, namely, the data that is measured from the process and the products. Data collection has become popular since the automation of plants and the introduction of various quality systems. Proper utilization of this vast source of information could enable *con-*

*tinuous learning* in the plants and whole enterprises. In the food and bioprocess industry of today, the data, however, is rarely used for this purpose.

Continuous learning should first of all ensure that past mistakes are avoided, e.g. that products are of a constant, high, quality. In addition, it could enable more flexible production management by optimizing the operations whenever changes in the raw materials, production equipment or product demand happen. Individual recipes can be constructed for batches that are performed in unusual vessels or with somewhat different raw materials. The possibility to schedule such batches and still obtain reasonable product quality allows for significant improvements in overall plant efficiency. Oppose this to typical current practice, where data is collected but recipes are frozen, i.e. some variation is accepted at the cost of product quality variations or stringent raw material specifications.

Advances in computing, both in the computing power and methods, have brought new possibilities for utilizing the production data. A particularly promising approach for this is *case-based reasoning*, CBR [12, 14]. In CBR terminology, every batch of a process (or any incident, e.g. a quality deviation or an alarm, in a continuous process) is a *case* that can be learned from. The combination of experience captured in such cases and existing process knowledge promises to enable effective planning of the process, and indeed of the production.

A case-based reasoner works by adapting and reusing solutions to previous problems as opposed to constructing them from scratch [10]. As solved problems are added to the case memory, case-based reasoners gain experience. This allows the systems to automatically tune themselves to different environments. Such an approach is fruitful in situations where it is not possible or feasible to generate a complete domain model, which rules out using pure model-based reasoning approaches.

This article describes the approach used in the Sophist system [2] to combine case-based reasoning and a semi-qualitative process model for the design and

<sup>1</sup>Present address: Nokia Networks, P.O. Box 370, FIN-00045 Nokia Group, Finland. E-mail: robert.aarts@nokia.com.

\*Corresponding author: Tel.: +358 9 456 78613; Fax: +358 9 455 2103; E-mail: juho.rousu@vtt.fi.

optimization of recipes in different bioprocesses. In such applications, a complete domain model is almost never available, but some knowledge on the process – albeit vague and imprecise – can be collected from literature and process operators. On the other hand, the day-to-day production is often measured accurately, at least variables that are most important to production economy.

The recipe planning domain is mostly defined by *continuous* features and the recipe parameters to be planned are mostly continuous set-points. Discrete planning steps deal with decisions of adding or removing some ingredient or some process step to the recipe. The ordering of process steps is seldom interesting. Typically there is only one correct order which is given to the planner as background knowledge. In this aspect, the recipe generation task is in the middle-ground between planning and parametric design.

The remainder of this article is organized as follows. First, the central recipe design mechanisms in Sophist system are introduced with the aid of an example of batch fermentation planning. In particular, we introduce a novel revision selection algorithm that improves the system's ability to learn. The adaptive behavior of the planner is then verified in an experiment in the fermentation planning domain. Finally, the merits and the potential of the CBR approach to production management are discussed.

## 2. Bioprocess recipe planning by Sophist

Food and biotechnical processes are challenging targets for optimization. The processes involve complex raw materials of biological origin and biologically active agents such as enzymes or microbes. The quality of the raw materials vary from harvesting season to another and also somewhat during one season. If a constant, high, product quality is to be achieved, variations in raw material quality should be compensated for in the process. This makes "one-shot optimization" inadequate. Instead, a plant should be always prepared to adapt the process when variations are observed. As the course of the process is very much defined by the recipe, the obvious solution is to make recipe changes, but it is often difficult to choose between alternative adaptations: a change can correct the problem at hand but simultaneously introduce new problems. Knowledge-based systems can be used to overcome these difficulties as demonstrated by Aarts [1].

Sophist [2] is a recipe planning system designed specifically for bioprocesses. It suggests recipe changes to compensate for changes in the raw materials, in the required processing time, or even in the product specification. The proposed changes are based on previously collected process data and background knowledge of the process. In this section we use the planning of a batch fermentation recipe to illustrate how Sophist uses previous experience to suggest finely tuned solutions to new problems.

The most important component of the system (Fig. 1) is the case-based reasoning machinery which is used both for suggesting good starting points for recipe generation and for evaluating the generated recipes. Each case adds to the experience of the system and can be used to solve problems in the future. The cases in Sophist contain descriptions of the initial situation (e.g. ingredients), actions (recipe) and the outcome (product quality and resource usage). Table 1 depicts two cases from the batch fermentation planning domain.

Another main component is the semi-qualitative process model that is used to generate adaptations and to predict their effects. The model is inherently incomplete and is not capable of generating complete recipes from scratch. Instead, it relies on cases as starting points, *templates*.

The mode of planning in Sophist is offline. The system is intended to be used when the production plan for the plant is made. It communicates with the production database through an SQL interface. Hence, it is independent of the automation system that is used in the plant.

To start the recipe planning, the system fetches a set of similar cases to the current problem at hand. A revision of each one of those template cases is generated for the current problem by a recipe generation algorithm. The adaptations are based on the differences between the template and the new problem and potential pitfalls of the template. This recipe generation process, in turn, is encapsulated in a revision selection algorithm that checks the proposed plan against other similar cases to find out whether the solution is likely to lead to a good outcome.

When the selected recipe has been executed the outcome of the case is analyzed and the case is added to the casebase. Thus the system is able to learn from experience.

### 2.1. Selecting template cases

The recipe planning in Sophist starts by searching for non-problematic cases similar to the current problem at

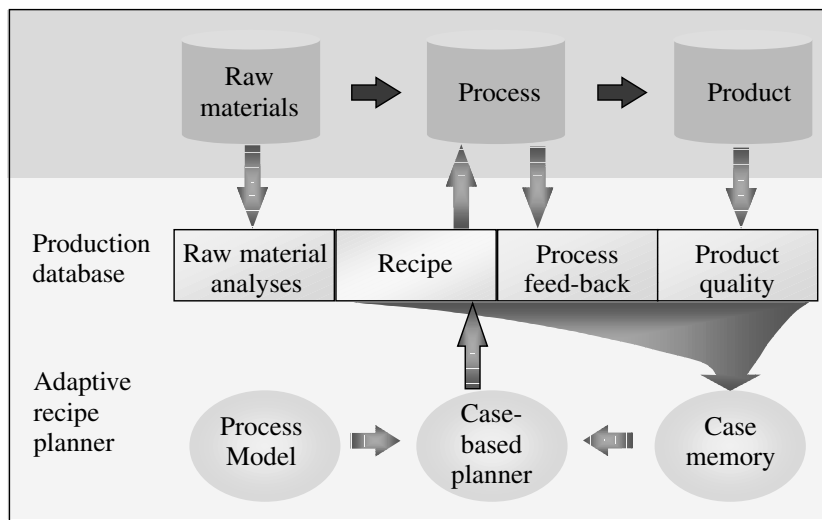


Fig. 1. An overview of the recipe planning architecture of Sophist.

Table 1

A template case, SC23, and the target case, SC29. The recipe for SC29 is adapted from the recipe for SC23

	SC23	SC29
Ingredients		
yeast MB	2.2	2.4
wort gravity	10.9	10.9
Recipe		
temperature	13	13
aeration	10.8	7.1
pitching	5	7.2
Outcome		
adf	79.4	79.4 (aim 80)
ethanol yield	73	81 (aim 85)
yeast yield	230	160 (aim 150)

hand. The similarity calculation is supported by a tree-structured index (Fig. 2). A separate subtree is allocated for ingredients, specification and the outcome. The last level of internal nodes in the index is labeled by names of features (e.g. measurements) that can be extracted from the cases. This organization allows us to flexibly include and exclude features from the calculation. For example, we may want to look for cases where the raw materials were similar irrespective of the product that was made.

The leaves of the index correspond to subsets of feature values. For example, the fraction of dead cells in the yeast is one property measured from fermentation cases. Separate index leaves are defined for cases where this fraction is below 5%, 5–10% or over 10%. The division of the range of the measurement is preferably given by hand. However, the system can also make crude division to index leaves based on the range and

accuracy of the measurement. The subsets are defined by fuzzy sets in order to make the similarity metric smooth.

Each leaf of the index is linked to cases that activate it. The links support weighting but no weight learning [18] is currently used. In Fig. 2, two cases, SC23 and SC29 (see Table 1), fully activate the fuzzy set “good” and do not activate other children of the “yeast MB” measurement. Thus, both cases are linked to the “good” index leaf and not to the other ones.

The index supports case retrieval as follows. The new problem is set as the active case in the casebase. This involves calculating an activity level for each leaf according to the fuzzy membership of the corresponding feature value found in the active case. Next, the pairwise similarity of other cases to the active case are calculated simply by summing up the activations of those index leaves that are linked to the case.

## 2.2. Recipe generation

Sophist uses a semi-qualitative process model [3] to support the analysis and adaptation of the template case. The primary motivation of this type of a model is the qualitative process theory by Forbus [9]. A process model in Sophist is a graph of *process* and *variable* nodes that are connected by weighted edges, *influences*. For instance the process “yeast growth” influences the variable “yeast in suspension”; the concentration of yeast will increase if the process is active. The activity of processes is conditional: “ethanol formation” takes only place when “temperature” is “ethanol forming” (see Fig. 2).

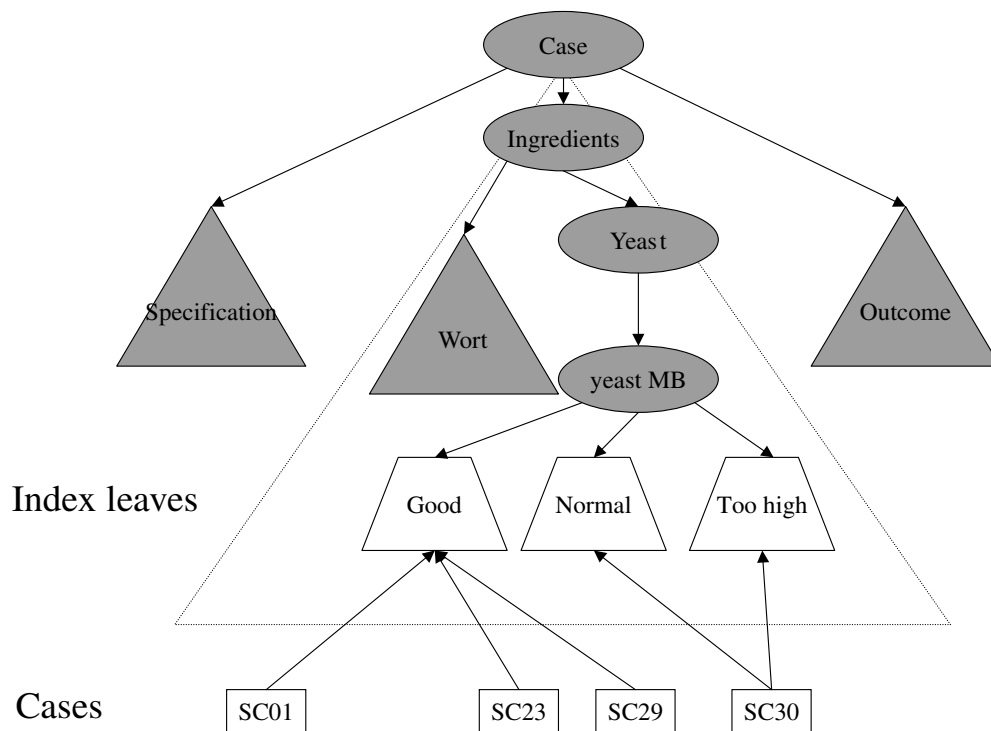


Fig. 2. The structure of the case index with Ingredients-subtree partially expanded. The links between a case and an index leaf represent a non-zero membership to the fuzzy set in question.

Now, assume that the task of the planner is to construct a recipe for a new case SC29 (see Table 1) which has a product specification and ingredients but no recipe yet, and also that a case SC23 has been submitted as the starting point, template case, of recipe generation.

The analysis of the template case SC23 starts with checking if the outcome of the template case would be suitable for the target. The planner notices two potential *pitfalls*, “an excess of yeast yield” and “a deficit of ethanol yield”, which indicate that if the recipe of SC23 was used for SC29 as such, too much yeast and too little ethanol may be produced. Each pitfall receives an importance score, which approximately quantifies the relative distance between the goal and the observation.

Next, the *case differences* are constructed. In the example, one of the constructed case differences is “a greater value for yeast MB”, which indicates that the yeast in SC29 is in a somewhat worse condition than the yeast used in SC23. Another case difference “a greater requirement for ethanol yield” denotes that a higher aim to ethanol yield has been set for SC29. As for pitfalls, an importance score is calculated for the case differences from the relative difference between the two compared values.

*Adaptation goals* are generated from the pitfalls and case differences by following the influence links in the model. For example, according to the model ethanol yield can be improved by increasing the activity of the process “pitching” or decreasing “yeast growth”. Similarly, from the model it is seen that “yeast MB” measurement affects negatively “yeast growth”. As SC29 has greater value for this measurement, this will generate the adaptation goal to increase yeast growth to compensate for the difference.

The weight of the influence link affects the importance of the adaptation goal: the weaker the effect of the process, the stronger adaptation goal is generated. Similar – or contradicting – adaptation goals are then combined: the adaptation goal “increase of yeast growth” disappears in this process.

Next, *adaptations* satisfying the adaptation goals are sought for from the model by looking at the controllable variables that, directly or via other processes and variables, affect the processes. In the example, “pitch rate” and “aeration rate” are found to influence “pitching” and “yeast growth”, respectively. Operations that are associated with the controllable variables are then asked for an adaptation that meets the adaptation goal. For example, the adaptation returned for “an increase

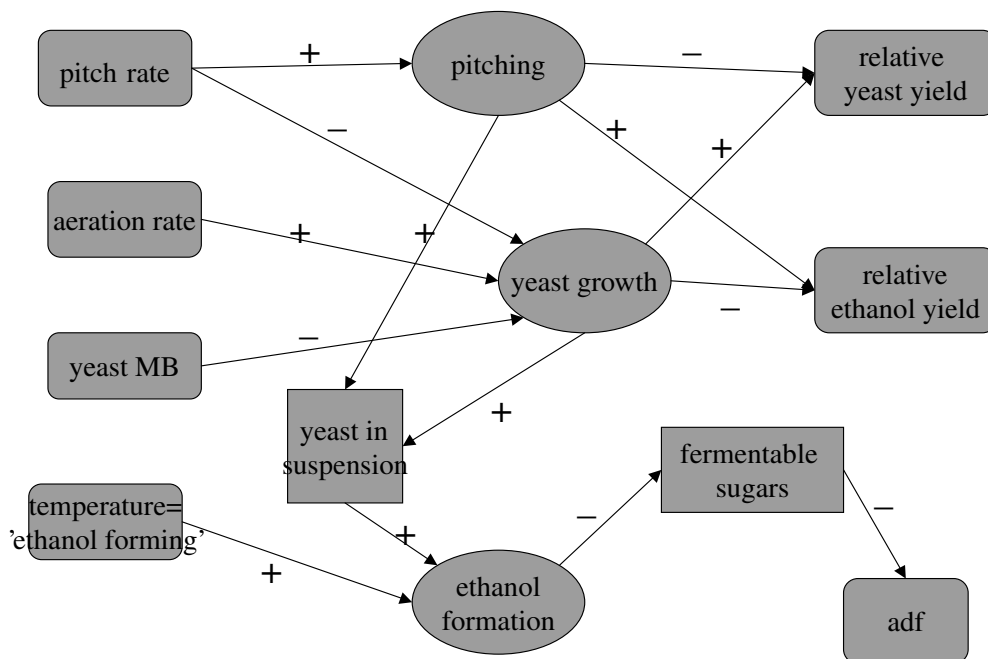


Fig. 3. A part of the beer fermentation model. The ovals represent processes, rectangles represent process variables and rounded rectangles represent measurements.

of pitching” is “an increase in the rate of the pitching step”. If no operation is found to help with the adaptation goal, the planner generates such an operation and adds it into plan. The effects of the adaptation, including side effects, are found by propagating the proposed adaptation back through the model. For example, it is found out that increasing pitchrate has a side-effect that accelerates ethanol formation and thus makes the process reach higher apparent degree of fermentation in the same time. As adaptation goals, also effects are approximately quantified by the model. Thus, each effect receives a score that can be directly compared to the importance scores of the adaptations.

Finally, the adaptations are ranked by their benefit ratio, the ratio of estimated desired effects to all effects of the adaptation, and the most effective adaptations are made. In the example, the planner decides to decrease the aeration rate and increase the pitching rate. The residual importance, or the degree of *dissatisfaction*, is calculated by subtracting the expected benefit from the importance of the adaptation goals. The resulting recipe is returned to the further analysis of the revision selection algorithm, described below.

### 2.3. The revision selection algorithm

The revision selection algorithm (Fig. 4) is basically a loop where the recipe generation algorithm is called

for each template case and selection criteria are applied to choose the most promising revision.

The selection criterion is two-fold. First, it is required that the adaptations in the revision do not bring the adapted recipe closer to a case different from the revision’s template case. This requirement is set in order to avoid oscillation; without it the recipe planning can get easily stuck in those cases where no satisfying recipes have been found yet.

As second criterion, the *degree of dissatisfaction* of the revision’s adaptations is used. It is calculated by the recipe generation algorithm by subtracting from the importance scores of the adaptation goals the corresponding effect scores. Thus, it is greater in those cases where the recipe adaptation algorithm was not able to find good enough adaptations, either due to the incompleteness of the adaptation algorithm or truly unattainable goals.

The requirement of keeping close to the template case can be interpreted as the cases dividing up the feature space into regions where only the “owner” may suggest recipes. Because each case will be asked for a recipe in turn, ignoring recipes that “invade” the neighborhood of another case will not do any harm; the owner case will have a chance to make a suggestion as well.

The planning strategy used is conservative in two ways. First, the planner resorts to a recipe of the closest

**Input:** A set of template cases  $c_1, \dots, c_k$  and a new case  $c^*$ . Case  $c_i$  a triple  $(s_i, p_i, o_i)$ ,  $s_i$  is the description of the initial situation,  $p_i$  is the plan and  $o_i$  is the outcome or goal. The new case is a triple  $c^* = (s^*, \emptyset, g^*)$ , having an empty plan,

**Output:** a plan for the new case

**Method:**

```

 $p_{best} \leftarrow p_1.$ 
 $bestCost \leftarrow dissatisfaction(s^*, p_{best}, g^*)$ 
for  $i \in \{1, \dots, k\}$  do
   $p_i^* \leftarrow generateRevision(c^*, c_i)$ 
   $c_i^* \leftarrow (s^*, p_i^*, g^*)$ 
   $cost \leftarrow dissatisfaction(c_i^*)$ 
   $distance \leftarrow \min\{d(c_i^*, c_j) \mid j \in 1, \dots, k\}.$ 
  if  $distance \geq d(c_i^*, c_i)$  and  $cost \leq bestCost$  then
     $p_{best} \leftarrow p_i^*$ 
     $bestCost \leftarrow cost$ 
fi
od
return  $p_{best}$ 

```

Fig. 4. The revision selection algorithm.

template case, if no good adaptations are found. Second, the model is not ultimately trusted if empirical evidence given by the cases disagree with the model. In that case, the suggested adaptations ultimately will violate the requirement of staying close to template and, as a consequence, no adaptations will be done. The conservatism slows down the convergence of the planner but avoiding bad adaptations is more important in industrial recipe planning.

In our earlier work [16], the idea of minimal adaptations was present as well; an adaptation cost was calculated based on the degree of dissatisfaction of the adaptation and the distance between the original and the adapted plan. However, this approach was vulnerable to falling into a local minima, because the planner could propose the same plan over and over again if no better template cases emerged during the planning. In other words, the method could not learn from its mistakes very well.

The requirement of the adapted solution being close to the original plan implements the idea that the template case is capturing some regularity in the problem domain. If the solution adapted for the current problem is closer to some other case than the template, then it might be so that the template's regularities are not present in the current problem. This approach bears some philosophical resemblance to the methods used by Ram and Santamaria [15] for case-based robotic control. Our implementation of these ideas, however, is quite different from theirs.

#### 2.4. Computational complexity

The time-complexity of the recipe planning is mainly affected by the case retrieval and adaptation generation.

The retrieval cost is incurred in the pair-wise similarity calculation and finding the  $k$  most similar cases. This involves scanning through the casebase and summing up the activation of the index leaves for each case. This takes time  $O(N|L|)$  in worst case, where  $N$  is the number of cases in the casebase and  $L$  is the set of the leaves in the index. After calculating the similarities, the set of  $k$  most similar cases are selected. Currently, this is done rather inefficiently, namely by sorting the cases. This process takes  $O(N \log N)$  time. Thus, the total time-complexity of the case retrieval is  $O(N(|L| + \log N))$ .

The time-complexity of the adaptation generation is  $O(|E||C| + |p||F|)$  where  $E$  is the set of edges in the process model graph,  $C$  is the set of case features,  $p$  is the set of operations in the template plan and  $F$  is the set of process nodes in the graph. The term  $|E||C|$  follows from the fact that the whole model graph may have to be traversed for each case difference and there can be at most one case difference per feature. The second term follows from noticing that the number of adaptation goals remaining after canceling contradicting goals may not exceed the number of process nodes in the graph. For each adaptation goal, each operation of the template plan needs to be checked to find possible adaptations.

Since adaptations are generated for  $k$  most similar cases, the total time-complexity of revision selection phase is  $O(k(|E||C| + P|F|))$ , where  $P$  is the maximum number of operations in the template recipes. Typically,  $|F|$  and  $P$  are small ( $< 10$ ),  $|C| < |L|$  and only a small part of edges in  $|E|$  are traversed. Therefore, in practice, retrieval dominates adaptation in terms of computational complexity.

### 2.5. Possible improvements to the planning scheme

The current recipe planning machinery in Sophist relies on two main ideas: a single template case is the starting point of recipe adaptation and the process to be controlled is described by a semi-qualitative process model. The fact that a single case is picked as the starting point certainly makes the methodology easier to comprehend by people not having mathematical education. However, a different strategy where a few neighboring cases take part in the recipe generation phase could work better in some situations, for example, dealing with very noisy domains. The biggest difficulty in implementing such a scheme is the integration to the process model. A voting or averaging approach can be considered: the recipe generation phase is run for a few *surrounding* cases [19] of the new case. An average of the generated recipes is computed, weighting with the inverse of the adaptation distance and the dissatisfaction estimate [17].

The use of the semi-qualitative process model is another possible area of improvement. In its current form the dynamic nature of the process is not well modeled. That is, the model does not make clear that, for example, lengthening fermentation time after a point where all fermentable sugars are consumed will not rise the degree of fermentation. In Sophist's planning scheme such knowledge comes into play indirectly: an unnecessarily long fermentation time is penalized in the template selection phase. However, it would clearly be more efficient to avoid these problems already in recipe generation.

Representing dynamic effects would require some changes to the process modeling syntax. At this time, the model only describes the relations between variables *averaged* over the whole process. A dynamic qualitative model would also need to describe how the relations change over the course of the process. Such relations are handled in, for example, in the qualitative reasoning formalism of Kuipers [13].

Finally, the idea of evaluating the recipes with cases described in the previous section could be taken a step

further. Currently, whole recipe is discarded if the result is too close to another case. This can be seen as an extreme form of case-based evaluation [12]: a solution is accepted or rejected as a whole depending what is the closest case to it. Performing fine-grained reasoning about the quality of the near case in the spirit of HYPO system [4] could prove beneficial.

More importantly, some case-repair strategy should be developed to improve the systems chances to find well-working solutions. Sophist already contains the necessary components to implement a *model-based* repair strategy: First, there often is one or two adaptations that can be recognized as the cause of making the evaluation fail, that is, as being responsible of bringing the new case too close to some other case. Second, typically in the recipe generation phase a number of adaptations are discarded as superfluous. Thus, these adaptations never make it to the evaluation phase even though some of them could have passed the evaluation. Clearly, better integration of the adaptation commitment phase and the evaluation phase could improve the planner's chances to find a good recipe and, also, to save computational resources. It is possible to consider a case-based repair strategy as well, along the lines of the CLAVIER system [11], where the surrogate adaptations are found from cases.

## 3. Experiments

The described planning approach was tested in a fermentation planning task. The goal of the planner was to find a recipe containing the aeration rate, temperature and pitching rate (the amount of yeast put in fermentation) in order to satisfy the goals for process time and relative ethanol and yeast yields (per fermented sugars). The initial situation is described by the condition of the yeast and by the gravity (sugar content) of the wort to be fermented (see Table 1). The process time was given as a hard constraint. Thus, a slow fermentation resulted in a lower degree of fermentation in the end of the process.

To start the experiment, we generated around 30 planning cases that differed from each other in the initial situation (yeast condition) and goal (apparent degree of fermentation, ethanol yield, yeast yield). An *ad-hoc* recipe was selected for each case and the outcome was simulated with a simple differential equation model. The initial casebase thus generated was given to Sophist as starting point.

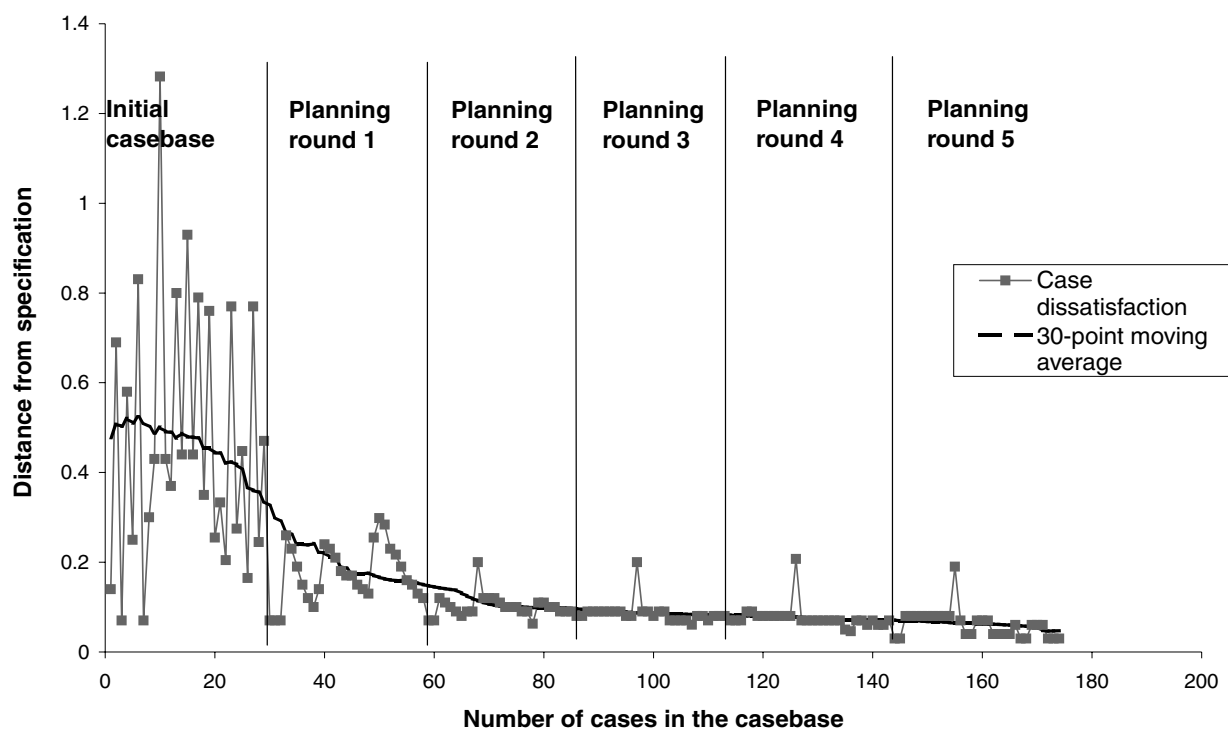


Fig. 5. The convergence of the recipes towards the optima. The  $x$ -axis represents the size of the casebase. Along the  $y$ -axis the case-wise degrees of dissatisfaction and a 30 point moving average are drawn.

In the first planning experiment, Sophist iteratively constructed a recipe for each planning problem, executed the recipe in a simulated fermentor, recorded the outcome and stored the newly completed case into the casebase for further use. After recipes had been planned for all of the problems, a next round was started with the same problems. This process was iterated five times or until the recipe converged.

The convergence of the recipes towards the optima is shown in Fig. 5. The graphs depict the distance from the specification (the sum of distances to the individual goals) of each case and a 30-point moving average. Most progress takes place during the first round. Note also that the solutions get better on average during a planning round and not only between successive rounds. The reason for this is that Sophist sometimes picks a recipe generated on the same round for a different problem and adapts it as needed. This phenomenon repeats in approximately the same way in each round, which explains the visually evident correlation between the error curves in successive rounds. One of the problems was significantly more difficult than the others and a good recipe was not found for it.

In our second experiment we tested what happens when new problems are entered to the casebase.

We picked six cases that had well-converged recipes, changed the specifications somewhat and let Sophist find new recipes for them. In Fig. 6 the evolution of the case outcome is shown for one of the problems. In this case, the goal for ethanol yield was risen and the yeast yield lowered. As in the first experiment, most of the progress took place on the first planning round after the changed specification was shown to the planner. The recipe adaptations made by Sophist are depicted in Fig. 7. The final goal is met by increasing pitching and lowering the aeration level. Sophist was able to find good recipes for the five other problems as well.

All in all, Sophist was able to find an optimized recipe for most of the problems that were presented to it. However, the simulation model used was very simple and did not contain noise, which of course is present in real bioprocess applications. Furthermore, in a real-world application the number of constraints will be higher than in these experiments. For example, in a beer fermentation task the flavor needs to be kept stable as well. These aspects are investigated in pilot fermentations that are currently underway.

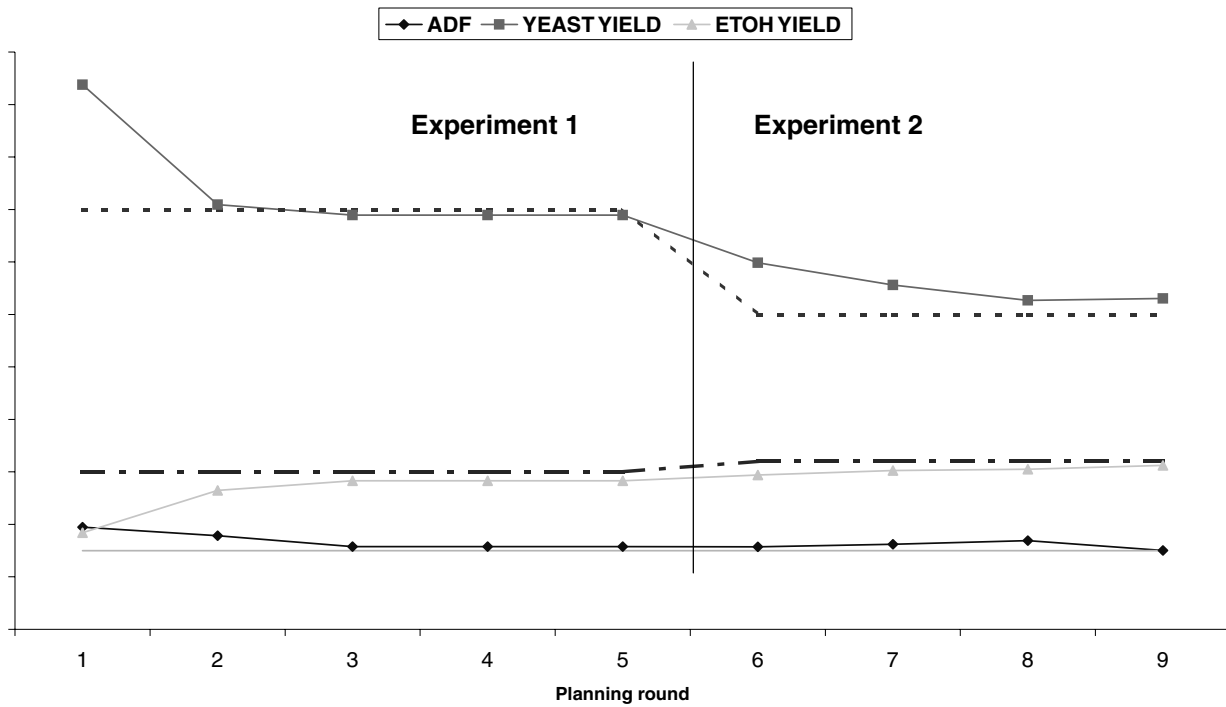


Fig. 6. The evolution of the case outcome during the planning experiments. The re-adaptation after a specification change requires a few rounds.

#### 4. Discussion

The recipe planning approach presented in this article combines two sources of information, namely the production data and the background knowledge. The idea behind such integration is, of course, to use the best parts of both to achieve something that is capable of more than the what is possible by using one knowledge source. In the case of Sophist, CBR and a semi-qualitative model were chosen but other implementation methods are possible as well.

CBR approaches are well-suited for solving problems where most of the time relatively minor modifications are needed (see, e.g. [7]). In such situations, constructing a solution from scratch may be considered wasteful or even risky. Moreover, a complete and accurate process model is needed for recipe generation from first-principles. Constructing such models for bioprocesses is a considerable, often prohibitively expensive, effort. A case-based planner can rely on weaker models such as (local) linear [5] or qualitative process models [8]. Successful application of case-based reasoning relies on the fact that the domain model is sufficiently complete locally, i.e. that the consequences of slight modifications can be known with sufficient certainty.

Perhaps the most important aspect of CBR is that the utilization of rich, qualitative, domain knowledge is

possible. The use this knowledge in Sophist allows for flexible analysis of process data. The system can use the model to reason not only with but also about cases. In addition, the model decreases the need for large casebases to start with. Naturally, when the casebase is small or when the problem distribution changes, recipe generation will rely more on the model, but as the casebase grows any shortcomings of the model will be offset by good examples in the form of cases.

Sophist has been developed in an application project. Hence, the main focus has been to develop an usable system rather than go into theoretical analysis of the methods. In particular, we have not tried to quantify the benefit Sophist has from using a model instead of relying purely on cases. Also, the way of integrating the domain knowledge into cases is certainly just one of the possibilities; In Section 2.5 we already discussed some alternatives.

One further aspect of the system to be noted is that the process model itself is static and not adaptive. Thus, the adaptivity of the system is only realized by the accumulation of the experience to the casebase that allows both finding good starting points for recipe generation and selecting most promising revisions among several candidates. Whether the approach used in Sophist would benefit from also trying to adapt the model is an another open question.

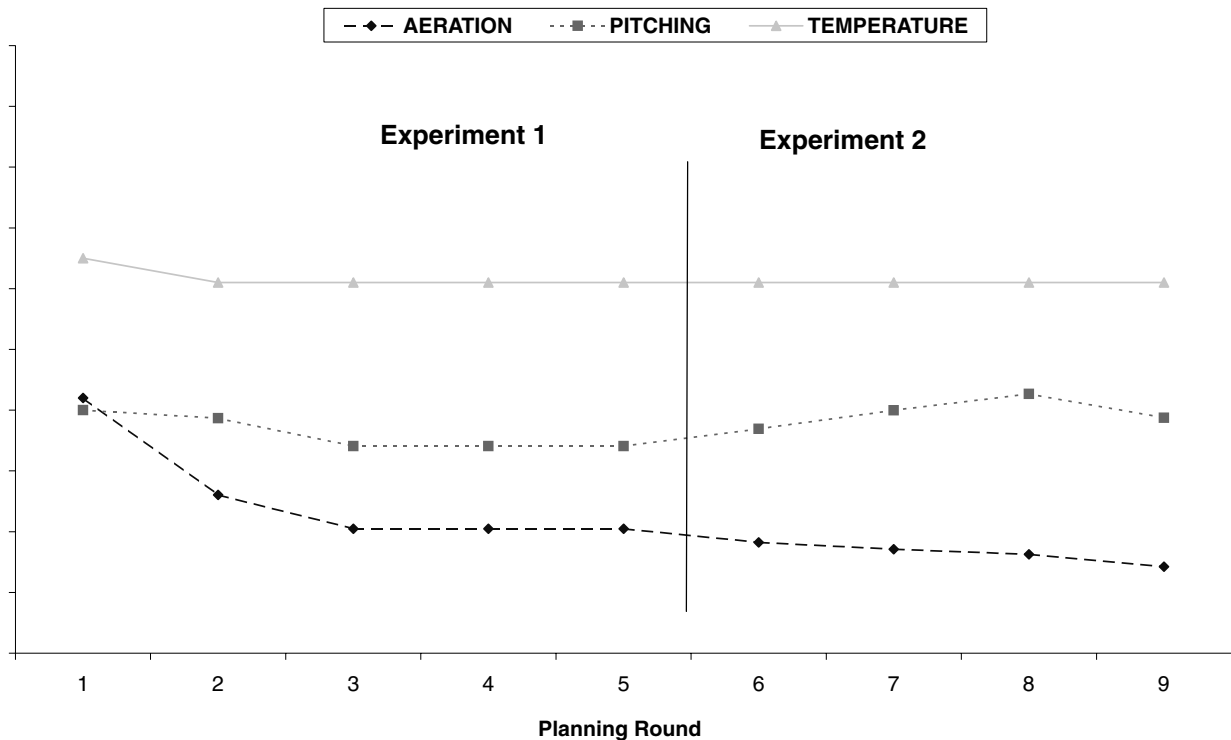


Fig. 7. The evolution of the recipe parameters during the experiment. The system mostly adjusts the levels of aeration and pitching to satisfy the goals.

The integrated approach of Sophist seems to fit best situations where enough qualitative knowledge is available to construct a model. New problem domains where such knowledge is not available will naturally suit this hybrid scheme badly, pure machine learning methods are more justified there. Looking at the other end of the spectrum, domains where complete and accurate models are available, neither CBR nor any other learning methods are needed. It seems, however, that the middle-ground where both background knowledge and learning can be utilized, is wide.

One of the strongest arguments in favor of CBR is the resemblance to human mode of reasoning: people tend to adapt solutions that have worked well in the past. In particular, this applies to process operators in the industry. Thus, the CBR approach may lower the psychological hurdle to accept knowledge-based systems in the industry.

The use of case-based reasoning implements the necessary elements for continuous learning [6]. Cases neatly pack all relevant data for a particular problem, and the system then allows searching for similarities. Note that it is not only possible to search for cases similar to a new batch, but also for example to cases where something similar went wrong, or even where

something different went wrong but for similar reasons, etc. The ever-growing casebase becomes an invaluable source of process knowledge.

However, a case-based planner, just as *any* production management technique, relies on the information that is collected from the production process. Automation of the plants has made data collection feasible, yet in all too many places vital data is not collected into a database; some crucial values are obtained manually and written on a piece of paper at best. Monitoring the quality of raw materials is even rarer; plants typically rely on the analyses that come from the supplier by fax or on paper together with the delivery. It is evident that companies that invest into creating and maintaining this IT infrastructure will have a competitive advantage in future.

Ensuring that all relevant data is stored on a computer in a structured manner is a big step forward towards *total quality management* (TQM) and continuous learning, and is the first step in the deployment of Sophist. Sophist itself is implemented as a framework that can rapidly be specialized for different processes. Currently, the mash planning application [2] of Sophist is in test use in two breweries. The fermentation plan-

ning application described in this article is in pilot scale tests.

## 5. Conclusions

Case-based reasoning, as demonstrated by Sophist, can be a powerful tool for effective process and production management. It fills a gap between traditional production planning and process control. A purpose made framework such as Sophist can be deployed especially rapidly. Moreover, a case-based planner effectively implements continuous learning of process knowledge. However, to do so, it is important that quality management is taken seriously so that all relevant data is recorded appropriately and made accessible to the computer system.

## Acknowledgements

Funding by Oy Panimolaboratorio (PBL) and by the National Technology Agency in Finland (TEKES) is gratefully acknowledged.

## References

- [1] R.J. Aarts, *Knowledge-based systems for bioprocesses*, Technical Research Centre of Finland, Publications 120, 1992.
- [2] R.J. Aarts and J. Rousu, Towards CBR for bioprocess planning, *Lecture Notes in Artificial Intelligence* **1186** (1996), 16–27.
- [3] R.J. Aarts and J. Rousu, Qualitative Knowledge to Support Reasoning About Cases, *Lecture Notes in Artificial Intelligence* **1266** (1997), 489–498.
- [4] K.D. Ashley and E.L. Rissland, A case-based approach to modeling legal expertise, *IEEE Expert* **3** (1988), 70–77.
- [5] C.G. Atkeson, A.W. Moore and S. Schaal, Locally Weighted Learning for Control, *Artificial Intelligence Review* **11** (1997), 75–113.
- [6] J.M. Barr and R.V. Magaldi, Corporate Knowledge Management for the Millennium, *Lecture Notes in Artificial Intelligence* **1186** (1996), 487–496.
- [7] W. Cheetham and J. Graf, Case-based reasoning in color matching, *Lecture Notes in Artificial Intelligence* **1266** (1997), 1–12.
- [8] G.F. DeJong, Learning to plan in continuous domains, *Artificial Intelligence* **65** (1994), 71–141.
- [9] K.D. Forbus, Qualitative Process Theory, *Artificial Intelligence* **24** (1984), 85–168.
- [10] K. Hammond, Explaining and Repairing Plans That Fail, *Artificial Intelligence* **45** (1990), 173–228.
- [11] D. Hennessy and D. Hinkle, Applying case-based reasoning to autoclave loading, *IEEE Expert* **7** (1992), 21–26.
- [12] J. Kolodner, *Case-Based Reasoning*, Morgan-Kaufmann Publishers, Inc., San Mateo, CA, 1993.
- [13] B. Kuipers, *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press, Cambridge, MA, 1994.
- [14] D. Leake, *Case-Based Reasoning: Experiences, Lessons and Future Directions*, AAAI Press, Menlo Park, CA, 1996.
- [15] A. Ram and J.C. Santamaria, Continuous case-based reasoning, *Artificial Intelligence* **90** (1996), 25–77.
- [16] J. Rousu and R.J. Aarts, Adaptation Cost as a Criterion for Solution Evaluation, *Lecture Notes in Artificial Intelligence* **1186** (1996), 354–361.
- [17] H. Toivonen, *Personal communication*, 1999.
- [18] D. Wettschereck, D. Aha and T. Mohri, A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning algorithms, *Artificial Intelligence Review* **11** (1997), 273–314.
- [19] J. Zhang, Y. Yim and J. Yang, Intelligent Selection of Instances for Prediction Functions in Lazy Learning Algorithms, *Artificial Intelligence Review* **11** (1997), 175–191.