Protótipo de Aplicação Para Otimização do Processo de Produção de Cerveja

Caroline L. Oliveira[1], **Eduardo F. Cordeiro**[1], **Carlos Andres Ferrero**[1]

[1]Instituto Federal de Santa Catarina – Campus Lages (IFSC)
Rua Heitor Villa Lobos, 222 – 88.506-400 – Lages – SC – Brasil

carolinelara35@gmail.com, eduardof.cord@gmail.com

andres.ferrero@ifsc.edu.br

Abstract. *In the productive scope it is common to encounter problems related to the optimization of a process and/or a product. Thus, a lack of a good schedule of the tasks in beer production was observed.To his end, a prototype of scheduling of tasks was developed through a Web Application, using the Pyschedule library in order to generate a scheduling optimization through a sequencing of tasks for the brewing productions. The prototype presented satisfactory results within the scenarios proposed by the application, showing that for smaller productions it is possible to obtain an optimal solution.*

Resumo. *No âmbito produtivo é comum encontrar problemas relacionados à otimização de um processo e/ou um produto. Dessa forma, foi observado uma falta de uma programação de agendamento das tarefas na produção de cerveja. Para isso, desenvolveu-se um protótipo de agendamento das tarefas através de uma Aplicação Web, utilizando a biblioteca Pyschedule com objetivo de gerar uma otimização de agendamento por meio de um sequenciamento de tarefas para as produções cervejeiras. O protótipo apresentou resultados satisfatórios dentro dos cenários propostos à aplicação, mostrando que para produções menores é possível obter uma ótima solução.*

1. Introdução

O âmbito cervejeiro possui uma forte presença e importância no mercado, sendo o Brasil apontado como o 3º maior fabricante de cerveja do mundo, tornando a cerveja a bebida alcoólica mais consumida no Brasil (Martins et al., 2018).

A cerveja continua sendo o produto mais registrado no Ministério da Agricultura, Pecuária e Abastecimento (MAPA, 2020), alcançando o número de 9.950 registros, bem a frente do segundo lugar, polpa de fruta com 2.535. Junto com o crescimento de registros pode ser projetado o aumento no número de cervejarias espalhadas pelo Brasil para os próximos anos com base no ritmo dos últimos períodos. Nos últimos vinte anos, observa-se uma taxa média de crescimento de 19%, nos últimos dez anos de 26% e, nos últimos cinco anos, de 36%, mostrando assim uma forte crescente no mercado e setor cervejeiro para o futuro, aumentando a relevância de pesquisas científicas que possam contribuir com o desenvolvimento desta área (Cordeiro, 2020).

Um problema presente nas áreas produtivas das cervejarias está relacionado com a eficiência e produtividade por conta da falta de uma programação ótima de agendamento

das tarefas de produção, como é o caso das etapas de fermentação e maturação pertencentes na área fria do processo cervejeiro (Baldo et al., 2014). A área fria pode ser definida como a etapa final do processo de produção da cerveja, antes que ela vá para o engarrafamento, possuindo uma rede de vários tanques onde a cerveja fica distribuída por alguns dias, passando por vários subprocessos. A falta de uma boa programação nesta etapa gera uma quantidade significativa de horas de ociosidade de tarefas, influenciando em vários indicadores de qualidade da cerveja de forma negativa, como o próprio sabor final que será entregue ao consumidor. Ociosidade de tarefas acontecem quando certa etapa produtiva, como a fermentação fica á espera para começar sua função que já deveria ter sido iniciada há algum tempo e este tipo de situação ocorre quando o equipamento necessário para tal tarefa está sendo ocupado ou bloqueado por outra tarefa que acabou fugindo do planejamento da fábrica.

Segundo dados fornecidos por uma cervejaria do sul do Brasil, as fábricas de cerveja chegam a trabalhar a 30% abaixo de sua capacidade máxima de produção, principalmente devido à falta de otimização da linha de produção. Por exemplo, se uma cervejaria foi projetada para produzir 100 hectolitros por semana (10 toneladas de cerveja) pode ser que esteja atualmente produzindo 70 hectolitros, o que impacta negativamente no lucro da empresa. Diante do exposto, neste trabalho é proposto o desenvolvimento de uma solução computacional para agendamento e otimização do processo de produção de cerveja, usando métodos matemáticos e de inteligência artificial.

Tem-se como objetivo principal deste trabalho de conclusão de curso, desenvolver um protótipo de solução computacional para a otimização e o agendamento de tarefas do processo produtivo de cervejas. Para alcançar esse objetivo tem-se os seguintes objetivos específicos:

- Levantar com os responsáveis do setor produtivo de uma fábrica de cervejas o processo de produção de cervejas, o ambiente de fabricação e as variáveis envolvidas no processo.
- Pesquisar métodos e ferramentas para a otimização de tarefas em processos produtivos;
- Desenvolver uma aplicação Web que permita ao usuário definir o volume de que se deseja produzir, executar o processo de otimização, e visualizar o agendamento das tarefas de produção.

Com o desenvolvimento deste trabalho espera-se contribuir: na previsão dos recursos que serão utilizados pela cervejaria no período de produção; traçar um caminho ótimo para cada lote dentro da produção desde o início até a sua finalização; diminuir o tempo gasto com o planejamento e o agendamento da produção; diminuir o tempo de produção para cada lote dentro de cervejarias.

A metodologia para desenvolvimento deste trabalho foi dividida em 8 etapas. Na Etapa 1, de revisão bibliográfica, foi realizada uma pesquisa no *Google Scholar*, em que foi definida uma *query* de pesquisa com o objetivo de encontrar trabalhos científicos que se assemelham ao conteúdo abordado neste projeto de trabalho de conclusão de curso. Na Etapa 2, de entendimento do processo cervejeiro, foi realizado o estudo do processo cervejeiro e realizadas reuniões com trabalhadores da área produtiva de uma fábrica de cerveja da região de Lages SC. Na Etapa 3, de levantamento de requisitos, é realizada a coleta e descrição de requisitos funcionais, não funcionais e regras de negócio da solução

computacional. Nesta etapa também são coletadas informações específicas sobre o processo de produção de cerveja como os tempos dos processos e subprocessos. Na Etapa 4, de prova de conceito, é desenvolvido um protótipo simplificado de otimização do processo cervejeiro para verificar que uma solução computacional para este problema poderá ser desenvolvida com tecnologias disponíveis na literatura. Na Etapa 5, de projeto de solução, são desenvolvidos diagramas de casos de uso, de atividades e protótipos de telas da solução computacional. Na Etapa 6, de desenvolvimento da solução computacional, é utilizada uma linguagem de programação e bibliotecas de apoio, como as relacionadas à otimização de processos. Na Etapa 7, de teste e validação, a solução computacional será testada e validada pelos desenvolvedores e por especialistas do domínio. Na Etapa 8, de produção científica, pretende-se desenvolver artigos científicos para publicação em conferências ou periódicos de alcance nacional ou internacional.

A metodologia utilizada neste trabalho fará uso de uma pesquisa aplicada exploratória quantitativa participante, de forma a gerar conhecimentos partir da interação entre trabalhadores e membros das situações investigadas (Silva e Menezes, 2005).

O restante do trabalho está organizado da seguinte maneira: na Seção 2 são apresentados os conceitos básicos para entendimento deste trabalho, bem como a descrição e artigos similares encontrados na literatura; na Seção 3 é descrito o levantamento de requisitos, o projeto da solução e a implementação da solução computacional; e, na Seção 4, são apresentados a conclusão e os trabalhos futuros.

2. Fundamentação Teórica

Nesta seção são apresentados conceitos básicos e trabalhos relacionados, fundamentais para entendimento deste trabalho. Na seção 2.1, de Processo Cervejeiro, aborda-se de forma geral como ocorre o processo de produção de cerveja. Na seção 2.2, de Trabalhos Relacionados, serão mostradas algumas das fontes que inspiraram o desenvolvimento prático deste trabalho. Nas seção 2.3 e 2.4, serão ilustradas as principais ferramentas utilizadas neste projeto, sendo elas o *Pyschedule* e o *Dash*, juntamente com exemplos de implementação que demonstram o funcionamento de cada uma.

2.1. Processo Cervejeiro

Existem diversas formas para se produzir cerveja, dentre elas, duas formas de produção podem ser citadas: a artesanal e a industrial. Na forma artesanal, a cerveja é produzida de forma caseira por poucas pessoas com baixo orçamento em maquinários, utilizando poucos processos. Na industrial a cerveja é produzida em grandes volumes, com equipes de centenas de pessoas com um alto orçamento em maquinário, sendo essas informações retiradas de entrevistas realizadas com a equipe do setor financeiro da fábrica estudada em questão.

As etapas de produção e suas principais características que serão apresentadas estarão totalmente vinculadas e descritas de acordo com a capacidade de uma planta industrial. Nesse contexto, o processo clássico de produção de cerveja é formado pelas fases de: (1) Preparação do mosto, (2) fermentação, (3) centrifugação, (4) maturação e (5) filtração. Essas fases são apresentadas na Figura 1 e são descritas a seguir.

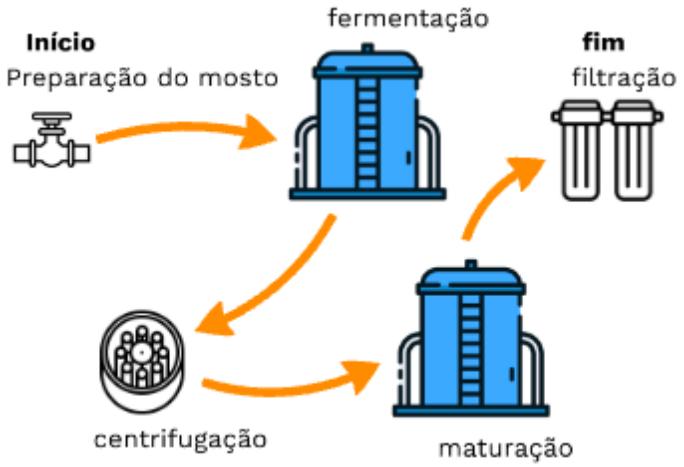

Figura 1. Planta do processo de produção de uma cerveja.

A Fase 1 consiste na preparação do mosto, que é feito adicionando-se ao malte: água e lúpulo; em seguida, leva-se o mosto à fervura. Logo após isso, adiciona-se a levedura e coloca-se a mistura para fermentar em tanques específicos para esse fim, dando início à Fase 2. Durante a fermentação retirasse o álcool e o gás carbônico a partir da quebra do açúcar contido no mosto, podendo ficar de 5 a 10 dias nesta fase. Após realizada a fermentação, na Fase 3, o mosto passa por uma centrífuga, que é uma etapa rápida com o objetivo de quebrar novamente os açúcares restantes. Na Fase 4 é realizada a maturação, em que a mistura ficará em repouso por dois dias, além de ser o momento ideal do processo para a adição de especiarias e outros ingredientes, melhorando os sabores e os aromas da receita. Na Fase 5, de filtragem, os restos de resquícios sólidos são completamente retirados da mistura, finalizando o processo da produção da cerveja.

Todo esse processo, levando em consideração seus tempos, ingredientes e etapas, muda de acordo com qual tipo de cerveja se deseja produzir. No caso de uma receita de cerveja sem álcool, deve ser acrescentada ainda uma Fase de Desalcoolização (retirada de álcool) entre a maturação e a filtragem.

Produzir uma cerveja num nível industrial requer alto custo de energia e água, o que exige cuidados quanto à reutilização de água e à redução da energia gasta, buscando sempre um maior volume de cerveja no fim da linha de produção. Um projeto adequado de otimização e agendamento das tarefas do projeto cervejeiro pode contribuir significativamente na melhoria dessa situação.

2.2. Trabalhos Relacionados

Para estabelecer uma pesquisa de trabalhos semelhantes a este projeto foi realizada uma revisão bibliográfica da literatura. A pesquisa foi realizada no Google Scholar utilizando uma *string* em inglês de consulta "*planning scheduling optimization brewing beer*". Observou-se que a pesquisa em inglês seria mais efetiva, pois o tema não é muito encontrado na língua portuguesa. Através do resultado da consulta foram selecionados os primeiros 20 artigos. Para cada um desses artigos selecionados foi realizada uma leitura dos resumos e objetivos dos trabalhos. Desta forma foi atribuído um valor de relevância,

de 1 a 5, para cada artigo, sendo 1 pouco relevante e 5 muito relevante. Dos 20 trabalhos selecionados, foram escolhidos 8 contendo as maiores notas, e desses foram escolhidos os 4 trabalhos de maior relevância e importância com o assunto que está sendo tratado neste projeto.

O trabalho proposto por Zheng et al. (2016) cita a utilização de um algoritmo de otimização para as cervejarias, levando em consideração o tempo gasto pelos especialistas na área para a solução do planejamento, além do desperdício com matéria prima, gerando menos volume no final do processo em comparação com o que realmente poderia ser produzido. Com isso, apresenta vários algoritmos existentes, porém entre eles é dado um foco especial ao algoritmo Colônia de formiga. Este artigo apresenta um novo tipo de Colonia de formiga, capaz de otimizar processos contínuos, o qual obteve sucesso ao ser testado num cenário real. Comparado com a versão clássica do algoritmo, o novo, obteve melhores resultados, diminuindo o tempo de processamento que antes era de 122 segundos, para 11 segundos, ao mesmo tempo que aumentou os nós de busca.

O trabalho de Goldman et al. (1997) *Honeywell batch scheduler*, um programa baseado em restrições de sistemas de agendamento para produções divididas em lotes. A tecnologia empregada foi o *Constraint Envelope Scheduling*. Obteve um sucesso considerável e serviu como fonte de dados para mostragem de pontos fortes e fracos sobre a tecnologia. Foi realizada uma comparação breve sobre processos contínuos com processos em lotes, ressaltando a limitação das plantas de fábrica em lotes, em relação às várias receitas a serem produzidas, utilizando os mesmos recursos ou até mais, gerando tarefas de limpeza entre trocas de receitas, além de poder mudar o tamanho do lote por receita aplicada e os tempos de processo necessários. Concluindo o artigo, foi citado todo o aprendizado técnico obtido em relação ao desenvolvimento da tecnologia.

O artigo de Sáenz-Alanís et al. (2016) apresenta um problema real sobre a falta de uma otimização de processos industriais de uma cervejaria localizada no México, e tem como objetivo o desenvolvimento de uma aplicação para realizar o agendamento e otimização destes processos de forma mais rápida para a cervejaria. Foi escolhido como área de atuação da aplicação, o cozinhamento do mosto e fermentação, sendo estas duas áreas as mais favoráveis para otimização levando em consideração suas complexidades de maquinário e atividades em paralelo. Para o desenvolvimento da aplicação, foi utilizado uma heurística conhecida como *Greedy Randomized Adaptive Search Procedure (GRASP)*, a qual favorece desenvolvimentos de aplicações para a parte de otimização de múltiplas atividades simultâneas. Ao concluir o projeto foi observado que, em 20 cenários de teste, a aplicação conseguiu diminuir o tempo de processo em 44% e o tempo de processamento para achar a solução em quase um dia comparado com o exercício humano que antes era o utilizado.

No artigo escrito por Shen e Smalov (2018) são apresentados três modelos de tomada de decisão no agendamento de tarefas do processo de produção de cerveja, Algoritmo Genético (GA), *Simulated Annealing* (SA) e Algoritmo de otimização da colônia de formigas (ACO). Através desses algorítimos são feitas comparações a fim de entender qual das ferramentas foi a que minimizou de forma eficiente o tempo total de produção. O problema a ser otimizado é do tipo *Job-shop problem (JSP)*, um problema de otimização nas quais as tarefas são atribuídas a recursos em momentos específicos e que não pode fornecer uma solução específica, é um problema bem complexo de ser solucionado.

Dentre os testes feitos a ferramenta que possui o melhor desempenho foi o Algoritmo Genético que executou melhor em comparação com o *Simulated Annealing* e o algoritmo de otimização da colônia de formigas, com desempenho de 35% sobre o valor sem otimização.

Após a leitura dos trabalhos relacionados observou-se que esses trabalhos não disponibilizaram o código fonte de suas implementações ou projetos em funcionamento de forma que pudéssemos adaptar ao nosso projeto e até com o objetivo comparar diferentes propostas. Por esse fato, neste trabalho será utilizada uma abordagem baseada em programação linear inteira, uma subárea da otimização combinatória computacional.

Os trabalhos serviram para um melhor entendimento sobre como transformar as variáveis e indicadores fabris para linguagem de código, abrindo possibilidades para diferentes formatos de como tratar o problema estudado neste artigo. Foi decidido colher esse aprendizado e tentar algo diferente e novo para este projeto, pois além de tudo, a complexidade da fábrica estudada foi levada em questão e a construção de algo mais robusto e personalizado se mostrou como necessidade.

2.3. Otimização Combinatória e a Biblioteca Pyschedule

Problemas de otimização combinatória são comuns no mundo real, principalmente na indústria, mas também em organizações públicas. Por exemplo, em universidades e institutos federais, no início de cada semestre é realizada a alocação das salas de aula e laboratórios de informática para as disciplinas do semestre, bem como a construção do quadro de horários dos docentes. Nesse tipo de problema, tem-se como entrada de dados os espaços disponíveis na instituição, as restrições de horário dos docentes e as disciplinas que devem ser ministradas pelos docentes. A saída do problema é uma solução viável que aloca os espaços para as disciplinas, de forma que os docentes não tenham dois espaços ou disciplinas alocados no mesmo horário. Em geral, procura-se não apenas uma solução viável, mas uma boa solução, que maximize ou minimize algum objetivo, como o grau de espalhamento do quadro de horários docente, podendo ser de interesse da instituição que a solução tenha preferência por alocar os horários dos docentes em dia contíguos, como de segunda-feira a quarta-feira, ou de quarta-feira a sexta-feira.

Além do problema supracitado, no cenário industrial é comum encontrar outros problemas de otimização, como o de roteamento para entregas de produtos, em que o objetivo é realizar todas as entregas com a melhor rota de visitação para os veículos da empresa, e o sequenciamento de tarefas, conhecido em inglês como *job-shop problem*, é um sistema em que a partir de conjunto de tarefas (*jobs*) de produção, um conjunto de máquinas, um conjunto restrições de execução de tarefas e um conjunto de alocação de máquinas são organizados para desenvolver uma agenda de otimização. O objetivo consiste em agendar as máquinas para executar as tarefas e concluir a produção no menor tempo possível. Esse tempo para concluir todas as tarefas é conhecido como *makespan* (Morales e Ronconi, 2016).

Em geral os problemas de otimização podem ser caracterizados da seguinte forma:

- Dados de entrada: dados essenciais para produzir a otimização, por exemplo, o volume que se deseja produzir de determinado produto.
- Soluções viáveis: um conjunto de valores para as variáveis do problema que satisfazem as restrições pré-definidas, por exemplo, uma solução que resolve o pro-

blema mas que não é necessariamente a melhor.
- Função objetivo: uma função que quantifica quão boa é uma solução e associa um valor a cada solução viável.
- Objetivo: encontrar a solução viável com o melhor valor. O melhor valor pode ser o menor (minimização), ao se considerar por exemplo o custo ou o tempo, ou maior, ao se considerar o lucro (maximização).

A biblioteca *PySchedule*, disponível para a linguagem de programação Python (Nonner, 2019), possui uma série de funcionalidades para modelar e resolver problemas de otimização combinatória. Existem outras bibliotecas de otimização combinatória para essa linguagem, como *IBM Decision Optimization*, *CP Optimizer Modeling* e *Google OR-Tools*, no entanto, *Pyschedule* fornece um conjunto de funcionalidades de mais alto nível, o que facilita a modelagem de problemas para o programador. Essa biblioteca está disponível publicamente sob a licença *Apache License* 2.0, que permite ser modificada, distribuída e usada comercialmente, inclusive em ambientes privados, mas ficando a garantia e responsabilidade por conta de quem a utilize. A biblioteca fornece suporte a:

- Relações de precedência: uma tarefa T_A deve ser executada antes de outra tarefa T_B.
- Custos de entrada: uma tarefa T_A só pode ser executada a partir de uma certa unidade de tempo após início da otimização, servindo como um atraso.
- Restrição de períodos: pode-se restringir períodos (unidades de tempo) sobre quando um recurso ou uma tarefa está disponível ou não para ser usado(a) pelo modelo.
- Alternativa de recurso: uma T_A pode ser executada pelo recurso R_X ou R_Y.
- Capacidades de recurso: um recurso R_X pode ser usado por um número definido de tarefas ou não pode executar tarefas de grupos diferentes, sem antes passar por uma tarefa intermediária.

Os elementos sobre otimização do processo cervejeiro mencionados são fundamentais no contexto deste trabalho de conclusão de curso. Como mencionado na Seção 2.1, a produção de cerveja apresenta as fases de fermentação, centrifugação e maturação, as quais possuem *relações de precedência*. Essas etapas podem ser executadas em tanques fermentadores, maturadores ou flex (suporte para maturar e fermentar), de acordo com os *requisitos de recurso*. Embora os tanques possam ser usados mais de uma vez para fermentar a mesma receita de cerveja, na necessidade de fermentar outra receita, é necessário tarefas extra de limpeza após cada etapa citada anteriormente, denominada de *Cleaning in place* (CIP), o que justifica o uso desta biblioteca com suporte para *capacidades de recurso*. Em alguns casos especiais, como a cerveja sem álcool, é necessário a adaptação de mais etapas de tarefas, como a própria desalcoolização da cerveja, realizada por uma máquina específica.

Para modelar um problema de otimização, *Pyschedule* disponibiliza um conjunto de classes, as quais são descritas a seguir:

- *Scenario* (cenário): representa um cenário de otimização, com a função de guardar e organizar as tarefas, os recursos e as restrições da otimização em questão. Possui parâmetros como:
  - *name*: nome ao cenário;
  - *horizon*: intervalo de tempo que se passa a otimização;

- *Task* (tarefa): representa uma tarefa e ficará responsável por alocar um recurso e um intervalo de tempo específico dentro do *horizon*, o tamanho deste intervalo equivale ao tamanho da tarefa (tempo de execução - início e fim). As tarefas aceitam como parâmetros:

  - *name*: nome da tarefa;
  - *length*: intervalo de ocupação de tempo da tarefa;
  - *delay cost*: custo a mais de tempo para cada *delay* de atraso para alocar a tarefa;
  - *my attribute*: possibilidade de criar um parâmetro customizado para ser adicionado nas restrições, facilitando e ampliando a otimização;

- *Tasks* (tarefa): representa uma lista de tarefas, normalmente usado para gerar um conjunto de tarefas similares. Além dos parâmetros de *Task*, também apresenta:

  - *is group*: valor booleano, informando se as tarefas da lista são emancipáveis (podem ser trocadas de posição durante a otimização) uma com as outras;
  - *num*: Quantidade de tarefas para serem criadas;

- *Resource* (recurso): representa um recurso, que será responsável por realizar uma ou mais das tarefas criadas para a otimização. Aceitando como parâmetro:

  - *name*: nome do recurso;
  - *cost per period*: adiciona uma penalidade ao recurso, para cada unidade de tempo que esse ficar locado, servindo muito bem para que o modelo utilize este recurso com menor frequência;

- *Resources* (recursos): representa uma lista de recursos, normalmente usado para gerar um conjunto de recursos similares, que serão responsáveis por realizar uma ou mais das tarefas criadas para a otimização. Aceitando como parâmetro:

  - *name*: nome dos recursos;
  - *num*: quantidade de recursos que se deseja criar. Normalmente ao utilizar esta chamada de método, é comum querer criar vários recursos de um mesmo tipo que praticam a mesma função;
  - *cost per period*: adiciona uma penalidade ao recurso, para cada unidade de tempo que esse ficar locado, servindo muito bem para que o modelo utilize este recurso com menor frequência;

No Algoritmo 1 é apresentado um exemplo da modelagem de um cenário em pequena escala de otimização usando *Pyschedule*, considerando a produção de 5 lotes de cerveja com três fases cada: fermentação, maturação e centrifugação. Nesse algoritmo, na linha 4 é criado o cenário S; nas linhas 7 a 11 são construídos os seguintes recursos: 3 fermentadores, 2 maturadores e 2 centrífugas; nas linhas 13 a 15 são inicializadas as listas de tarefas de fermentação, centrifugação e maturação; e, nas linhas de 17 a 22, são criadas as tarefas das 3 fases para os 6 lotes, totalizando 18 tarefas. Nessas tarefas a fermentação irá demorar 24 horas, a centrifugação 5 horas e a maturação 12 horas.

```
from pyschedule import Scenario
# Cenário de agendamento de 168 (24 horas em 7 dias)
# unidades de tempo (horas)
S = Scenario('Cervejaria', horizon=168)

# Definição dois Recursos para a otimização
R_fermentadores = S.Resources('Fermentador', num=3)
R_maturador_1 = S.Resource('Maturador_1')
R_maturador_2 = S.Resource('Maturador_2')
R_centrifuga_1 = S.Resource('Centrifuga_1')
R_centrifuga_2 = S.Resource('Centrifuga_2')

fermentacao = []
centrifugacao = []
maturacao = []

numero_de_cervejas = 6
producao = range(1, numero_de_cervejas + 1)
for lote in producao:
    fermentacao.append(S.Task('FC' + str(lote), length=24 ))
    centrifugacao.append(S.Task('CC' + str(lote), length=5))
    maturacao.append(S.Task('MC' + str(lote), length=12))
```

Algoritmo 1. Exemplo de cervejaria de produção em pequena escala.

2.3.1. Operações

Uma vez definidos o cenário, os recursos e as tarefas, é possível realizar operações, por meio de operadores sobrecarregados implementados na biblioteca, para relacionar tarefas aos recursos. A seguir são apresentados os principais operadores:

- Operador **+=** : Significa acrescentar, adicionar, alocar(fazer uso).
  - *tarefa fermentar* += *fermentador 1*;
  - *tarefa fermentar* += *maturador 4*;
- Operadores **alt** e | : Signfica *ou*.
  - *tarefa fermentar* += *alt(fermentador 1, fermentador 2)*;
  - *tarefa fermentar* += *fermentador 1* | *fermentador 2*;
- Operadores **<** e **>** : Descreve a ordem de acontecimento das tarefas.
  - *fermentar 2* > *fermentar 1* (tarefa fermentar 2 deve acontecer depois de fermentar 1, independente dos recursos utilizados);
  - *fermentar 2* >= *fermentar 1* (tarefa fermentar 2 deve acontecer assim que terminar fermentar 1, independente dos recursos utilizados);
  - *fermentar 2* < *fermentar 1* (tarefa fermentar 2 deve acontecer antes de fermentar 1, independente dos recursos utilizados);

No Algoritmo 2 é apresentado um exemplo de utilização dos operadores mencionados acima (**+=**), (**|**), (**>=**), (**>**), como continuação do exemplo de cervejaria de produção em pequena escala do Algoritmo 1.

Nesse exemplo são definidas as etapas de fermentar, maturar e centrifugar para cada um dos seus lotes de cerveja. Uma condição especial, comumente usada na produção

de cerveja, foi definida para os lotes 1 e 2, os quais serão centrifugados, nas máquinas centrífugas 1 e 3 respectivamente.

```
1  # Assignando recursos as suas tarefas utilizando Requisitos de recurso
2  for fermentar in fermentacao:
3      fermentar += alt(R_fermentadores)
4
5  for maturar in maturacao:
6      maturar += R_maturador_1 | R_maturador_2
7
8  for centrifugar in centrifugacao[2:]:
9      centrifugar += R_centrifuga_1 | R_centrifuga_2
10 centrifugacao[1] += R_centrifuga_1
11 centrifugacao[2] += R_centrifuga_2
12
13 # Aplicando regras de ordenamento por meio das Relações de precedência
14 for i,fermentar in enumerate(fermentacao):
15     for j, centrifugar in enumerate(centrifugacao):
16         if i == j:
17             S += centrifugar >= fermentar
18
19 for i,centrifugar in enumerate(centrifugacao):
20     for j, maturar in enumerate(maturacao):
21         if i == j:
22             S += maturar >= centrifugar
```

Algoritmo 2. Exemplo do uso de operadores do Pyschedule na modelagem de produção de cerveja.

O Algoritmo 2 inicia estabelecendo que cada tarefa de fermentação pode ser executada em algum dos fermentadores disponíveis (linhas 2 e 3). Posteriormente, nas linhas 5 e 6, define-se que a fase de maturação em um dos dois maturadores disponíveis; nas linhas 8 e 9, define-se a fase de centrifugação, que pode ocorrer em qualquer uma das centrífugas, exceto para a centrifugação dos lotes de cerveja 1 e 2, pois para estes, foi designada a centrífuga 1 e 2, respectivamente. Nas linhas 14 a 17, adiciona-se ao cenário as restrições de que, para fase de fermentação e centrifugação, se estas forem referentes ao mesmo lote, devem ocorrer de forma sequencial, isto é, a centrifugação deve iniciar em tempo maior ou igual ao fim da fermentação. Por fim, seguindo essa mesma ideia, nas linhas 19 a 22, são adicionadas as restrições de que as tarefas de maturação devem ser iniciadas ao terminar a etapa de centrifugação para cada lote de cerveja.

Uma vez modelado o problema e implementado em linguagem de programação é necessário procurar soluções viáveis utilizando um algoritmo de otimização. Como já mencionado, a biblioteca *PySchedule* utiliza diferentes algoritmos, denominados *solvers*, dentre os quais *CP Optimizer, Mixed Integer Programming (MIP, 2020), e OR-Tools*. No Algoritmo 3 é apresentado um exemplo da otimização de produção da cervejaria em pequena escala com o algoritmo MIP, como continuação do Algoritmo 2.

O Algoritmo 3 inicia definindo que a função objetivo é de minimizar a quantidade de tempo total para construir as tarefas; posteriormente, executa-se o processo de otimização do cenário S com o método MIP; e, por fim, mostra o gráfico com o resultado da otimização.

```
from pyschedule import solvers, plotters
# Assinando a função objetivo ao modelo
S.use_flowtime_objective()

# Busca uma solução para o problema do cenário S usando MIP
solvers.mip.solve(S)

# Mostra graficamente o resultado da otimização
plotters.matplotlib.plot(S)
```

Algoritmo 3. Utilização do algoritmo MIP para otimização da produção de cerveja em pequena escala.

A execução dos Algoritmos 1, 2 e 3 retorna como resultado o agendamento das tarefas, que é representado graficamente pela Figura 2. O objetivo de minimização foi alcançado de forma ótima, com um valor de *makespan* = *80* horas. Como essa solução é ótima não existe outro agendamento de tarefas que possa concluir todas as tarefas em menor tempo.

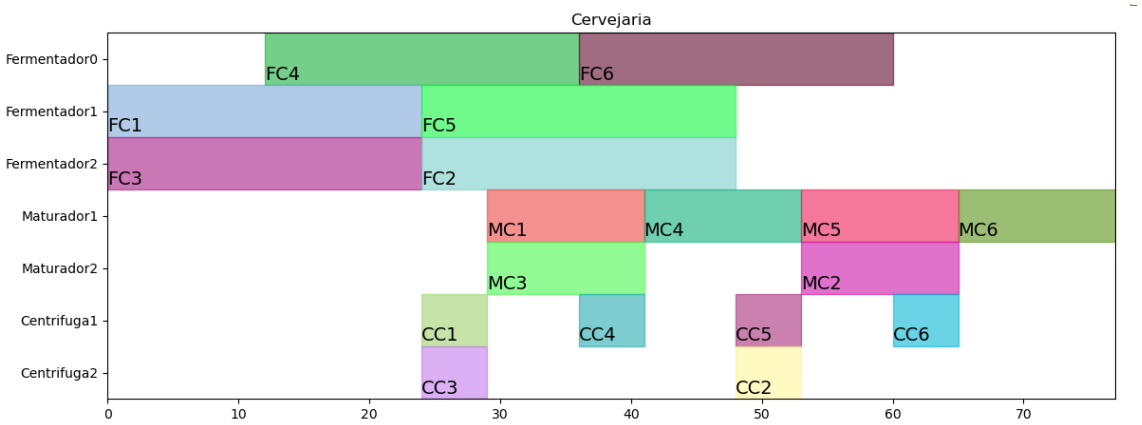

Figura 2. Visualização gráfica do agendamento de tarefas após a otimização.

Na Figura 2 observa-se o resultado da otimização na forma de um Diagrama de Gantt, onde no *eixo Y* são representados os recursos, no *eixo X* o tempo em que ocorre a execução das tarefas, e os retângulos representam as tarefas, em que a largura do retângulo representa a duração da tarefa.

Apesar da simplicidade do exemplo mencionado, a modelagem para um problema real de produção de cerveja é mais complicada, em função de que devem ser agendadas tarefas de produção de diferentes cervejas paralelamente, em maior volume, com tarefas de limpeza na linha de produção e dos tanques. Para se ter uma ideia, para produzir 280 mil hectolitros do produto em uma infraestrutura com 45 tanques de capacidade de 4 mil hectolitros cada é necessário fazer o agendamento de aproximadamente 550 tarefas no período de 30 dias.

2.4. Biblioteca Dash

A visualização de dados e a interação com o usuário são dois pontos importantes na construção de aplicações que envolvem otimização. Desta forma, buscou-se uma ferramenta de visualização de dados simples e completa, que possa ser facilmente integrada com o

resultado da otimização da biblioteca de otimização apresentada na Seção 2.3. Como a interação do usuário e dos dados são o ponto central da aplicação, a biblioteca Dash se mostrou adequada às aplicações desejadas, pois é um *framework python* criado para produzir aplicações web, baseada em ferramentas de interface como o *Flask, Plotly.js* e *React.js*.

Para o melhor entendimento da ferramenta, é preciso compreender o funcionamento da aplicações *Dash*. Uma aplicação usando *Dash* é composta de duas partes (Plotly, 2017): *layout* e *callback*.

A parte do *layout* é responsável pelos componentes visuais da aplicação, a ferramenta *Dash* fornece um conjunto de componentes através do *dash_core_components* e *dash_html_components*.

- *dash_core_components:* fornece classes para todas as tags HTML, por exemplo, quando usamos a tag *id* para atribuir o nome a um campo.
- *dash_html_components:* - gera componentes de nível superior.

Além desse conjunto de componentes o *Dash* também permite que sejam criados outros componentes com JavaScript e React.js. Uma definição mais simples de *layout* seria a de descrever aparência do aplicativo e uma árvore hierárquica de componentes. O algoritmo 4 mostra uma implementação simples do *layout*.

```
# Criamos um input que vai receber qualquer valor
app.layout = html.Div([
    html.Div(["Entrada: ",dcc.Input(id='my-input', value='valor inicial
        ', type='text')]),
    html.Br(),
    # Criamos um nome para esse componente
    html.Div(id='my-output'),
])
```

Algoritmo 4. Exemplo de layout simples.

A segunda parte da aplicação são os *callbacks* que são responsáveis pela interatividade do aplicativo. A ferramenta Dash usa funções de retorno de chamada, essas funções de chamada são funções *Python* que são acionadas pelo *Dash* automaticamente sempre que um componente de entrada é alterado. O *Dash* utiliza o decorator *@app.callback* para saber a propriedade de uma *entrada* e *saída*, assim, é possível fazer a alteração das propriedades. Além disso, o *Dash* fornece o componente *dash.dependencies* que é usado nas funções de chamadas nas propriedades de um componente.

- dash.dependencies - Propriedades de um componente
  - *dash.dependencies.Input:* propriedade de entrada
  - *dash.dependencies.Output:* propriedade de saída
  - *dash.dependencies.State:* estado da propriedade

O algoritmo 5 mostra uma implementação simples do *callback*.

```
# Criamos um decorator
# Ao escrever este decorador, pedimos ao Dash para chamar essa função
    sempre que o valor do componente "input" mudar para atualizar os
    filhos do componente "output"
@app.callback(
    Output(component_id='my-output',component_property='children'),
    [Input(component_id='my-input',component_property='value')])
    # Função python retorna o valor atualizado
    def update_output_div(input_value):
        return 'Saída: {}'.format(input_value)
```

Algoritmo 5. Exemplo de callback simples.

Ao juntar as duas partes da aplicação, observe que o algoritmo 4 cria os elementos visuais e apresenta um campo para preenchimento o qual o usuário pode alterar. Após isso, o algoritmo 5 identifica uma alteração no componente específico (my-input) de entrada do *layout* e, em seguida, chama uma função que retorna ao usuário alguma alteração no componente específico de saída (my-output).

Em sequencia são apresentadas as figuras que representam a aplicação em execução. A Figura 3 apresenta a aplicação no seu estágio inicial, a Figura 4 apresenta a aplicação recebendo os dados inseridos pelo usuário e a Figura 5 apresenta a saída dos dados inseridos pelo usuário.

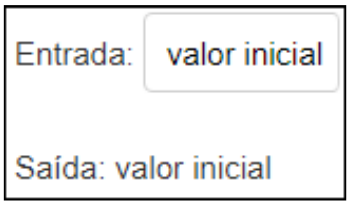

Figura 3. a.

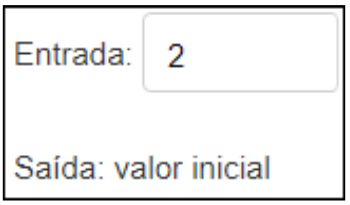

Figura 4. b.

Figura 5. c.

Em outras palavras, se o valor inicial da entrada for o valor inicial como mostra a Figura 3 no primeiro exemplo e o usuário alterar para *2* como mostra a Figura 4 no segundo exemplo, logo o valor de retorno será *2* como mostra a Figura 5 no terceiro exemplo. Todo o processo de alteração entre as duas partes da aplicação é feita de forma simples, assim, é possível ver a eficácia da ferramenta na análise e apresentação dos resultados em uma aplicação.

3. Desenvolvimento

Para o desenvolvimento deste projeto foi utilizado como base a prototipação de um ciclo do processo de desenvolvimento de software juntamente com a biblioteca Pyschedule e a biblioteca Dash. As etapas do desenvolvimento estão divididas em: Levantamento de Requisitos, Projeto da Solução, Implementação e Teste e Validação, todas essas etapas estão descritas a seguir.

Processo de Desenvolvimento de Software

O processo de desenvolvimento de software pode ser visto como um roteiro que determina as tarefas necessárias e a ordem que elas devem ser executadas para construir

um software de qualidade. Segundo Wazlawick (2013) em cada fase é definida as atividades, funções e responsabilidades. No processo de desenvolvimento de software, se fez necessário a utilização de uma ferramenta que auxilia e facilita este processo, com base no software implementado a ferramenta escolhida é o modelo de Prototipagem. A prototipagem visa o uso de protótipos para melhor entendimento do sistema, cada protótipo evolui até se tornar o sistema final.

A prototipagem é o ciclo vida que em toda versão evolui ou modifica seus requisitos e funcionalidades, além de revisar a estrutura do projeto a cada versão. A partir dos conceitos estudados em Sommerville (2016), entendeu-se que a prototipação pode ser dividida em etapas de evolução, em que cada etapa vai definir, adicionar, testar e validar a solução para o problema proposto. Este modelo é normalmente utilizado em cenários em que os requisitos obtidos são muito vagos ou não tão claros conforme menciona Sousa (2019).

A Figura 6 mostra um exemplo ciclo de prototipação na qual existem 6 etapas, sendo a *1 - Coleta e refinamento dos dados*, *2 - Projeto rápido*, *3 - Construção do protótipo*, *4 - Avaliação do protótipo pelo usuário*, *5 - Refinamento do protótipo* e *6 - Engenharia do Produto*. As 4 primeiras etapas são as principais, pois são nelas que se concentram a maior parte do trabalho, a primeira etapa é responsável por identificar e coletar os dados, a segunda etapa cria um esboço do problema através de interface, rascunhos, estrutura ou esboço. A terceira etapa coloca em prática a implementação da solução encontrada e a quarta etapa é responsável pela avaliação da solução implementada através de teste. A partir dessas 4 etapas estudadas, inicia-se a quinta etapa que analisa o resultado e refina o protótipo, como consequência é decido se vai para o processo de finalização do produto na sexta etapa ou redefine a solução do protótipo voltando para a segunda etapa, como mostra a seta no meio do ciclo. Normalmente, nesse tipo de modelo é comum a repetição do ciclo, dessa forma a solução pode ser melhor elaborada e testada.

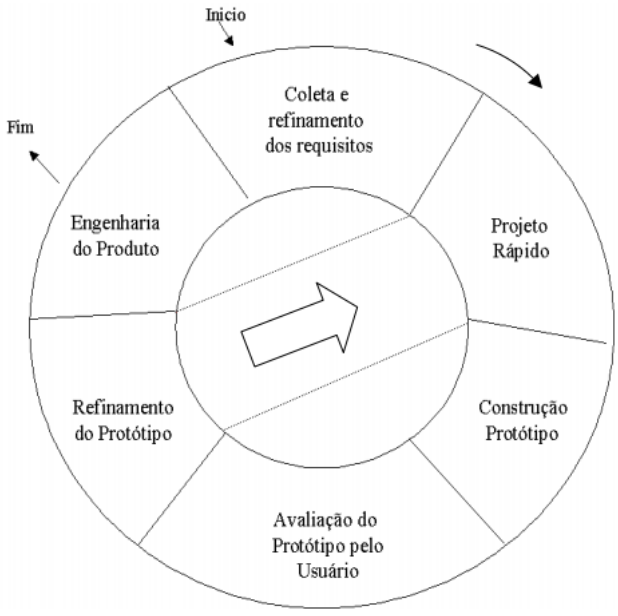

Figura 6. Visualização do ciclo de vida da prototipagem.

Para este projeto decidiu-se por reduzir as etapas do ciclo apresentado na Figura 6, pois entendeu-se que era necessário apenas as 4 primeiras etapas do ciclo, como mostra os conceitos estudados em Sommerville (2016). Dessa forma as principais etapas no processo de Prototipagem deste trabalho foram definidas para essas quatro etapas: Levantamento de Requisitos, Projeto, Implementação e Avaliação.

3.1. Levantamento de Requisitos

Para elaborar o levantamento de requisitos, deve-se entender o conceito de requisito a um projeto, só assim é possível coletar os dados corretamente. Um projeto possui requisitos e restrições, no qual os requisitos representam as necessidades e expectativas dos interessados, em geral o cliente. Por outro lado as restrições são fatores internos e externos com proposito de delimitar o projeto. Para definir requisitos e restrições é preciso entender o problema e coletar os dados, que pode ser feito de varias maneiras diferentes, como por exemplo entrevistas, questionários, prototipação, etc. É interessante mencionar que o levantamento de requisitos é muito importante no desenvolvimento do sistema.

Por meio da cervejaria que auxiliou na organização e orientou de forma clara os dados mais relevantes ao projeto e conforme a demanda do problema os requisitos foram levantados.

Exploração do Domínio: a partir do entendimento do problema através de pesquisas, identificar uma possível solução.

Definição de Requisitos: foram realizadas entrevistas com especialistas da produção de cervejaria.

Levantamento de Requisitos

Este etapa é a mais importante pois será feita a definição dos requisitos e das restrições que o software possuirá. A partir disso será traçado um esboço inicial para o protótipo.

Projeto

Um ponto de ênfase nesse etapa é a definição de funcionalidades que o software vai possuir, além da descrição de cada funcionalidade importante. Algumas funcionalidades podem ser implementadas pelos protótipos futuros, deixando a possibilidade de renovação a cada protótipo desenvolvido.

Implementação

Esta etapa define as prioridades de desenvolvimento, ressaltando o principal foco do desenvolvimento e mostrando alguns trechos de código que se fazem de grande importância na Implementação.

Avaliação

A etapa de avaliação visa testar com várias entradas e com várias situações que podem ocorrer no cotidiano e validar os resultados. Além disso, será feita uma descrição destes testes e a explicação do contexto de cada teste.

3.1.1. Exploração do Domínio

O formato de coleta de dados e requisitos foi realizado através de entrevistas com o pessoal da área de processos da cervejaria de uma fábrica de cervejas na região de Lages. As entrevistas foram feitas com o pessoal da equipe de operação, com os supervisores e gerentes da área.

Para compartilhar as informações coletadas, foi criado um Quadro demonstrando os maquinários de cada etapa de produção, utilizando como exemplo a produção de uma cerveja do tipo *Lager* (baixo tempo de fermentação), sendo as cervejas desse tipo as mais comuns entre os brasileiros

Quadro 1. Informações coletadas dos maquinários usados para produção de cerveja.

Planta - Área de Processos			
Nome	**Descrição**	**Tempo de CIP**[1]	**Duração**
Fermentadores	Tanques que servem para realizar as fases de fermentação no meio do processo produtivo da cerveja, podendo levar de 6 a 7 dias dependendo da cerveja.	6h	6 a 7 dias
Maturadores	Tanques que servem para realizar as fases de Maturação no meio do processo produtivo da cerveja. A maturação pode levar de 8 horas a 48 horas, depende da planta da área produtiva de cada cervejaria.	6h	48h
Centrifugas	Máquinas que servem para centrifugar a mistura da receita que sai da fermentação, enviando-a direto para a fase de maturação. Na centrifugação são retirados restos sólidos da mistura que prepara a cerveja.	3h	12h
Filtros	Presentes na filtração, normalmente é a ultima fase da produção de uma cerveja. Quando a mistura passa pelo filtro, ela sai completamente sem restos sólidos estando praticamente pronta para o empacotamento.	24h	10h
Linha de dosagem	Linha usada para transportar a cerveja para dentro dos fermentadores até o enchimento do tanque.	1h	12h
Linha de fermento	Linha usada para transportar o fermento para dentro e fora do fermentador, durante a fase de fermentação	1h	6h
Linha de maturação	Linha usada para transportar a mistura da cerveja para a filtração após a fase de maturação.	1h	-
Bombas de CIP	Servem para realizar o *Cleaning in place* (CIP). Tarefa que faz a limpeza dos maquinários durante a produção de uma cervejaria	-	-
Desalcoolizador	No caso de um receita de uma cerveja sem álcool, o desalcoolizador serve especificamente para esta tarefa de retirar o álcool da mistura	24h	12h

[1]Clean in Place é a higienização de partes internas de equipamentos, tubulações e tanques.

Com base nos dados coletados, entendeu-se que um dos problemas na fabricação de cervejas é que se leva mais tempo do que o necessário para sua produção seguindo o fluxo de de matéria prima e as condições do cenário. Neste projeto, com o auxílio de métodos matemáticos e computacionais, será possível associar às tarefas de produção de cerveja ao maquinário disponível, com o objetivo de reduzir o tempo de fabricação da cerveja e, consequentemente, o aumento da produção mensal.

3.1.2. Definição de Requisitos e Prioridade

Baseado nas informações coletadas através de entrevistas com pessoas especializadas na área da produção de cerveja foram definidos os requisitos para o desenvolvimento de um *dashboard*. Este *dashboard* contém as prioridades que foram estabelecidas conforme a demanda do problema, juntamente com pessoas especialistas no assunto. Para fazer a classificação das prioridades foi adicionada uma escala de 1 a 5, sendo 1 de menor prioridade e 5 de maior prioridade.

Requisitos Funcionais

Os requisitos funcionais são todas as funcionalidades ou características de um sistemas, de forma geral, como mostra Sommerville (2016), são ações que devem ser realizadas pelo sistema. Um exemplo de ação do usuário é adicionar uma cerveja à malha de produção, que consiste em adicionar uma cerveja ao processo do agendamento que será otimizado. Os Quadro 2 e 3 mostram os requisitos do projeto com a sua descrição e suas respectivas prioridades estabelecidas.

Quadro 2. Requisitos funcionais da tela do usuário para cadastro de lotes de cerveja para agendamento.

Requisitos Funcionais		
Id	**Descrição**	**Prioridade**
RF .A001	Todo agendamento terá um tempo determinado para ser otimizado. Podendo ser dias, semanas ou meses.	5
RF. A002	Para a otimização do agendamento, será usada uma lista de cervejas, onde será selecionada uma cerveja de cada vez.	5
RF. A003	Para cada cerveja selecionada, será adicionado um volume de produção, a quantidade de litros que se deseja produzir naquele determinado tempo de agendamento.	5
RF. A004	Após determinar a cerveja e o volume da mesma, ela será adicionada em uma lista de cervejas.	5
RF. A005	A cerveja pode ser removida da lista de cerveja.	3
RF. A006	A lista de cervejas pode ser excluída, caso queira iniciar uma nova lista de cervejas.	3
RF. A007	O usuário poderá solicitar o inicio do processo de otimização.	5

Quadro 3. Requisitos funcionais da otimização do processo de produção.

Requisitos Funcionais		
Id	**Descrição**	**Prioridade**
RF. B001	Realizar a otimização a partir de um arquivo de entrada com formato específico JavaScript Object Notation (Json), contendo a estrutura da cervejaria e os volumes para produção.	5
RF. B002	O arquivo de entrada para o modelo deve conter quais receitas devem ser produzidas, juntamente com a quantidade em hectolitros para cada receita	5
RF. B003	Permitir que tarefas possam ser executadas em apenas determinados recursos.	3
RF. B004	Permitir que tarefas possam ser executadas sequencialmente. Utilizando as regras de precedência.	1
RF. B005	Permitir quebrar a produção de cerveja em tarefas menores de acordo com a quantidade de recursos necessários para essas tarefas.	3
RF. B005	Dependendo do tipo de cerveja a ser produzido, o modelo deve ser capaz de passar o caminho de etapas correto para cada cerveja.	4

Diagrama de Caso de Uso

O diagrama de caso de uso, um dos diagramas *Unified Modeling Language (UML)*, apresentado na Figura 7 exibe as possíveis ações de serem tomadas pelos usuários, assim como as funcionalidades do sistema.

O diagrama da Figura 7 apresenta a diferenciação dos usuários se baseando nas restrições de acesso às funcionalidades da ferramenta de cada um. Cada caso de uso se insere dentro de uma raia definindo que a ação pode ser tomada pelos usuários conectados a ela. Assim, foram considerados dois usuários: supervisor e operador (lado esquerdo da Figura 7).

O supervisor é o responsável por todo o agendamento de horário para a produção. Por esse motivo tem acesso a todas funcionalidades (arraias de ações) de otimização e configuração da lista de cervejas, podendo acrescentar ou retirar qualquer item disponível da otimização, rodar o otimizador e gerar o resultado.

O operador é responsável por garantir a otimização da fábrica na prática. Por esse motivo, ele acaba tendo acesso bem mais restrito que o supervisor, se baseando apenas na visualização do resultado para conseguir colocar as atividades geradas em prática no seu dia a dia.

A função da ferramenta *rodar otimizador* no callback, chama o ator otimizador, para assim retornar o resultado. Outro ator existente no modelo de casos de uso da Figura 6 é o frontend, que tem por função, executar a ação de *Atualizar lista* quando houver alguma modificação na lista de produção cervejas para otimizar.

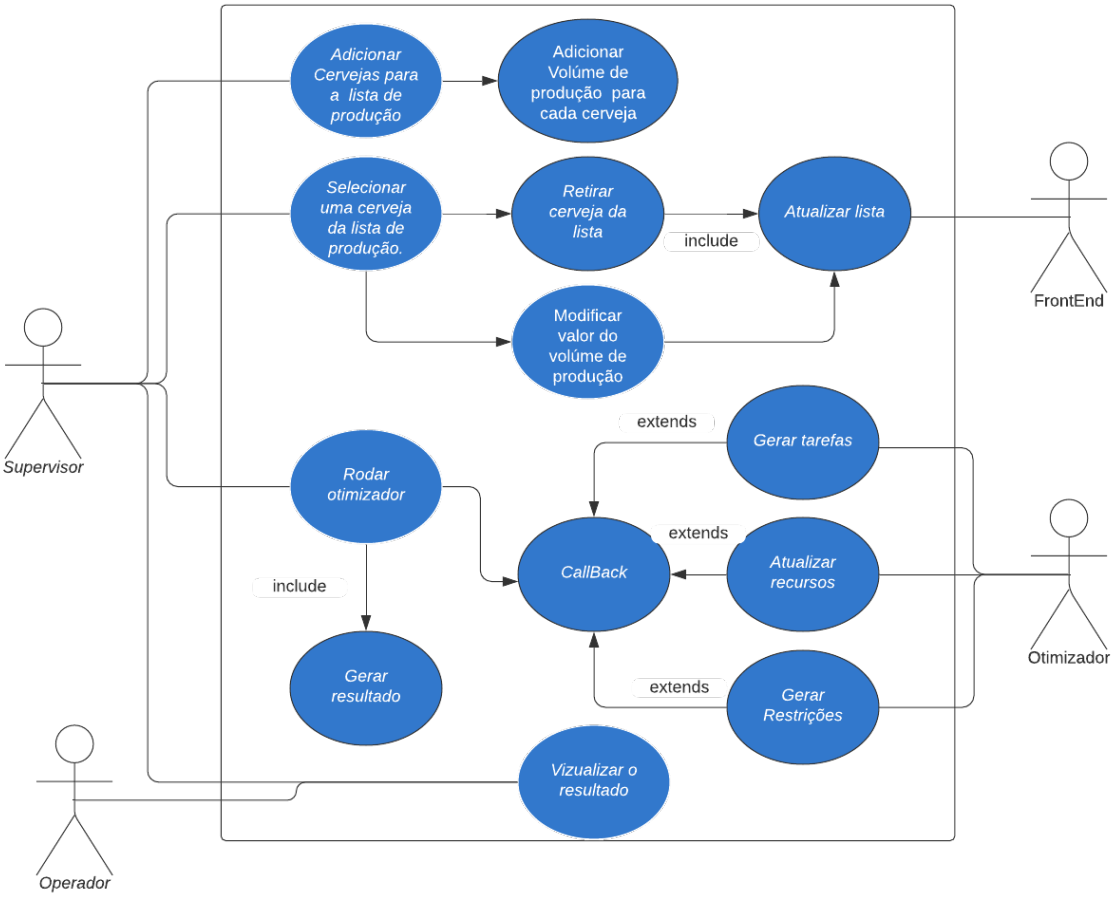

Figura 7. Diagrama de casos de uso UML, representando as possíveis ações de interação do usuário com a ferramenta.

Diagrama de Atividade

O diagrama de atividade da UML da Figura 7 descreve o fluxo de interação do usuário com a tela do cadastro do agendamento, sendo representados por círculos no diagrama o início e fim da interação do usuário, cada caso de uso se insere dentro de uma raia que define se a ação está sendo tomada pelo usuário, pela função de *callback*, dados ou otimização.

A Figura 8 é a representação do fluxo de interações das informações que o usuário disponibiliza através de suas ações na tela de cadastro do agendamento de cervejas. Para esclarecer o fluxo, dividiu-se em 3 raias, que representam camadas do sistema em contrapartida com as informações do usuário. A primeira raia retrata o *Supervisor* interagindo com a interface de agendamento de cervejas, é nessa raia que define-se o tempo de agendamento, seleciona-se as cervejas e seus respectivos volumes. Em seguida os dados da produção de cerveja são enviados para o otimizador. Posteriormente o usuário recebe o resultado da otimização e o agendamento de tarefas. A segunda raia é representada pelo *Callback*, que é a raia responsável pelas funções que interligam as camadas, é nessa camada que ficam as funções de conexão, é nela que liga-se as informações para que o usuário possa ver a lista de cerveja, adiciona-se uma cerveja e seu respectivo volume a uma lista de cervejas, define uma lista de dados que será enviada para outra raia e princi-

palmente verifica o status da otimização, que é nessa função que se detecta o resultado da solução. A terceira raia é retratada como *Otimização*, a mais importante camada, na qual é responsável por receber a lista de dados e otimizar, é nela que envia o status da otimização, cria as tarefas, lista os recursos e lista as restrições. Em seguida, essa raia vai otimizar e gerar identificadores de qualidade da solução, gerar a lista de tarefas agendadas, gerar gráficos de alocação de tarefas no tempo e por fim alterar o status de otimização. Posteriormente a esses processos será apresentado um resultado da otimização do agendamento na tela.

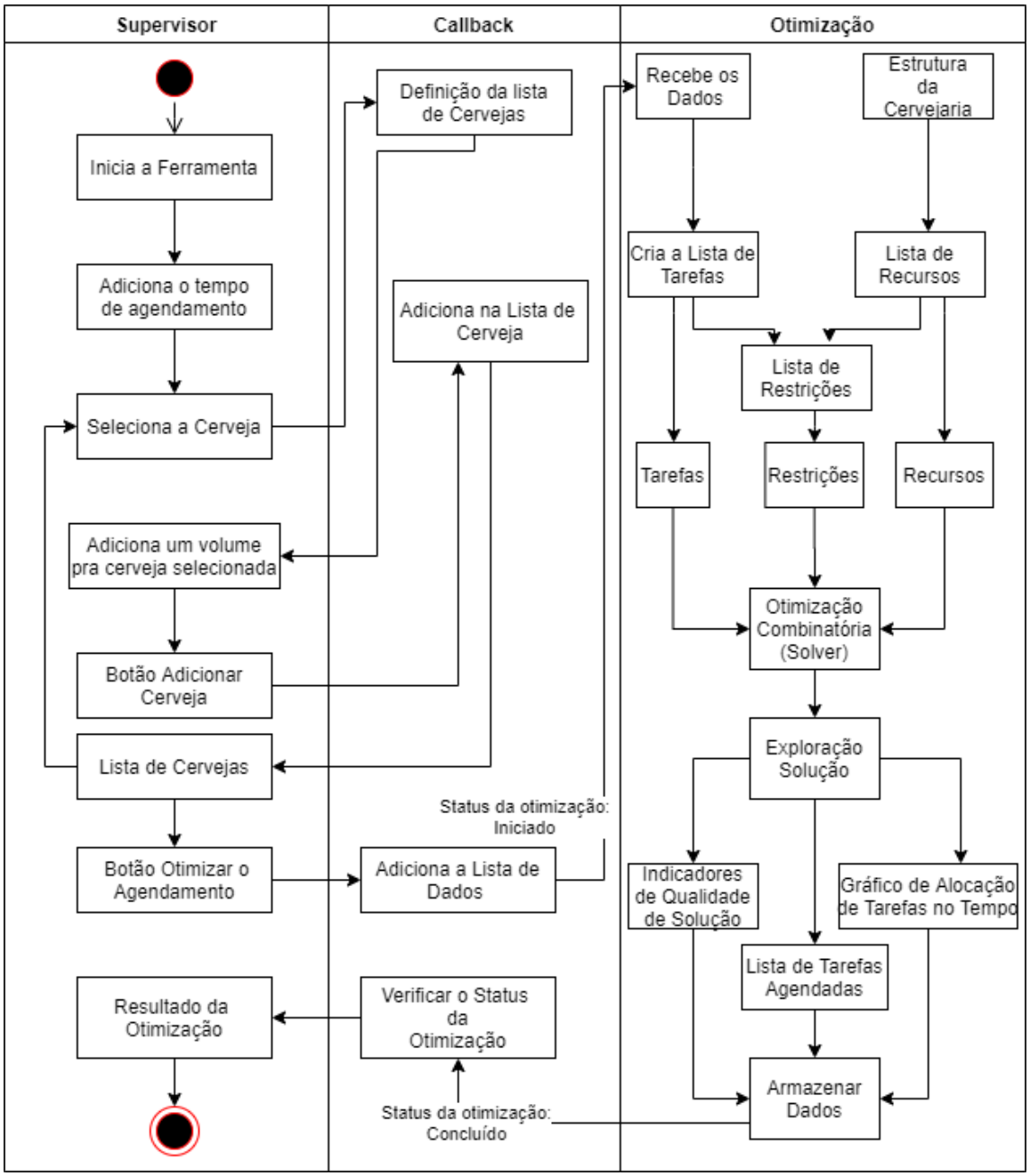

Figura 8. Diagrama de atividades da UML, representando o fluxo de interação do usuário na ferramenta.

Requisitos Não Funcionais

Os requisitos não funcionais são requisitos que não são diretamente relacionados com os serviços prestados pelo sistema aos seus usuários, segundo Sommerville (2016). Esses requisitos especificam restrições que o sistema deve obedecer. Um exemplo de requisito não funcional é o tempo de resposta do sistema. Além disso, a interface é bem intuitiva e fácil de ser utilizada, tem um bom desempenho e consegue entregar um resultado satisfatório. Um usuário novo não terá dificuldades para entender como o sistema funciona, com isto, nós adicionamos uma prioridade com peso 3 nesse requisito.

3.2. Projeto da Solução

A solução foi desenvolvida conforme os dados coletados e o problema levantado por especialistas na área de produção de cervejas. Para desenvolver o software foram utilizadas as técnicas de engenharia de software, por prototipação segundo as obras de Wazlawick (2013) e Sommerville (2016), que auxiliaram no desenvolvimento e evolução dos protótipos. A seguir tem-se um esboço inicial da ferramenta.

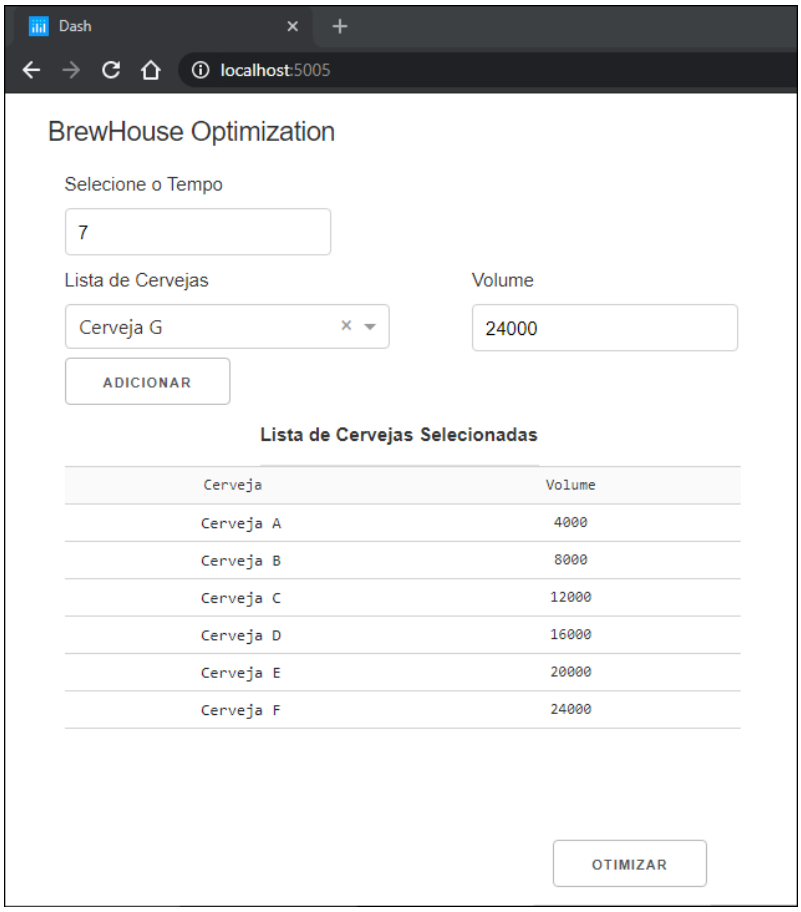

Figura 9. Tela de usuário: Cadastro de volume de produção.

Com o objetivo de melhorar o entendimento das ações do usuário, é interessante mencionar os componentes que compõem a Figura 9. A principio são 3 campos de entrada de valores, sendo o primeiro campo o do *Tempo* que será realizado o agendamento, no caso da imagem são 7 dias. O segundo campo possui uma *Lista de Cervejas*, no qual

acompanha do terceiro campo, que é o *Volume* desejado para ser fabricado dessa cerveja. Em seguida tem-se o primeiro botão *Adicionar* que adiciona a cerveja e o seu respectivo volume em uma *Lista de Cervejas Selecionadas*, esta lista é responsável por mostrar todas as cervejas que serão otimizadas. Após esse processo de adicionar cervejas, passa-se para o segundo botão. O segundo botão é o de *Otimizar*, que junta todas as informações passadas pelo usuário e envia os dados para o processo de otimização.

Como mostra a Figura 9, esta é a tela principal e o único meio de interação com o usuário, onde são adicionadas as informações de agendamento. Nesta tela, as informações são adicionadas e a partir disso, são processadas e otimizadas. Após a otimização das tarefas, retorna-se uma agenda de tarefas na tela principal.

3.3. Implementação

A implementação foi construída na base de algoritmos de alocação de tarefas e máquinas, conhecido como é *JobShop Problem*, dividida em duas partes, a primeira parte é o *Algoritmo de Otimização*, onde é realizada toda a estruturação dos dados e onde se dá inicio à construção das restrições de processo que darão estrutura à lógica do modelo e do projeto. Já na segunda parte que se dá o nome de *Ferramenta de Interface*, é realizada toda a interface para interação do usuário, coletando dados como: tipos de cervejas e volume de produção de cada uma. Esses dados são enviados para o *Algoritmo de Otimização* onde ele irá retornar como resultado um gráfico com todas as tarefas a serem realizadas para aquela otimização de processos da cervejaria.

3.3.1. Algoritmo de Otimização

Para começar a primeira fase de implementação foi necessário uma coleta bruta de dados sobre várias etapas do processo de produção de uma cerveja: Tempo de cada etapa; hectolitros mínimo e máximo necessários para produção; quantidade de etapas; possíveis horários de agendamento; Todos esses dados foram estruturados num formato de arquivo *.Json* para melhor gerenciamento destes e das restrições que foram criadas baseadas nestes dados coletados. Restrições essas que são em resumo, regras de produção da fábrica que precisam ser computadas em código para serem consideradas pelo modelo, servindo assim para otimizar o resultado de agendamento esperado.

```
1 # Realizando as Precedências das tarefas
2 tarefa_fermentar += alt(recursos.fermentadores)
3 tarefa_fermentar += tarefa_arriar*recursos.fermentadores
4 Cervejaria += tarefa_fermentar >= tarefa_arriar
```

Algoritmo 6. Exemplo da criação de restrições.

No Algoritmo 6 são apresentados os comandos necessários para criação de uma restrição de precedência: que consiste em atrelar os recursos necessários para tarefa fermentar na linha 2 e 3, para que possa ser feita a ordenação de precedência na linha 4, indicando que a tarefa fermentar deve ocorrer logo após a tarefa arriar. Este é um bom exemplo do quanto a biblioteca Pyschedule facilita o desenvolvimento e a implementação de restrições na modelagem de problemas do tipo *Job Shop Problem*.

O desenvolvimento da parte de otimização pode ser separado entre a criação das restrições como mostrada na Figura 6, mas também pela criação das tarefas que vão estar sujeitas a essas restrições e dos recursos que estarão executando essas tarefas, conforme mostra o Algoritmo 7.

```
1 def gerarTarefa_Arriamento(Cervejaria, tarefa, malha_produção):
2   for cerveja in malha_produção:
3     arriar = tarefa["arriar"]["tarefas"][cerveja["SIGLA"]]
4
5     nome = cerveja.nome
6     quantidade_tanques = cerveja.tanques
7     duracao = tarefaArriar.duracao
8
9     arriar = Cervejaria.Tasks(nome, quantidade_tanques, duracao)
```

Algoritmo 7. Exemplo da criação de tarefas.

3.3.2. Aplicação Web

Na segunda fase da implementação é importante entender o funcionamento da *Aplicação web* que possui duas partes, *layout* e *callback* . Essa divisão é necessária para que a interação do usuário flua de maneira correta. Para iniciar o desenvolvimento da implementação é necessário criar os *componentes visuais* que estão inseridos no *layout*, esses componentes são responsáveis por criar os campos, os botões, as tabelas, os gráficos, etc. Tudo que possa ser um *componente visual* será desenvolvido e implementado no *layout*.

A implementação do *layout* é composto de alguns campos de cadastro, pois a principal função da *Aplicação web* é cadastrar as informações necessárias para o *Algoritmo de Otimização*. Dessa forma, para cadastrar cervejas e criar um agendamento, são necessárias algumas informações:

- O tempo em que será feito o agendamento da otimização.
- A lista informando as cervejas disponíveis.
- O volume que cada cerveja vai possuir.
- Uma tabela que mostra as cervejas cadastradas com seus respectivos volumes.
- Um botão para otimizar o agendamento.

Essas são as informações necessárias para que a otimização ocorra e todos os dados mencionados sejam fornecidos pelo usuário. A seguir o algoritmo 8 mostra um exemplo de como as classes do *layout* funcionam. O algoritmo mostra a criação de uma *div* na linha 1 e em seguida na linha 2 a criação de uma tabela, a partir da linha 3 até a linha 14 são criadas as propriedades da tabela como na linha 3 o id *data-table* e duas colunas, uma coluna com o nome da *cerveja* e a outra coluna com o *volume*, após isso são adicionada algumas características visuais da tabela, como o número de linhas que a tabela possui em cada página, tamanho das células de cada linha, posição da escrita, tamanho da tabela, entre outras coisas . Nesse exemplo é possível entender como uma tabela é construída utilizando a *Biblioteca Dash* apresentada na Seção 2.4.

```
html.Div(
    dash_table.DataTable(
    id="data-table",
    columns=[{"name": "Cerveja", "id": "Beer"}] + [
    {"name": "Volume", "id": "Bulk"}],
    page_size=6,
    style_cell={'minWidth': 95, 'maxWidth': 95,
    'width': 95, 'textAlign': 'center'},
    style_table={'height': '250px'},
    style_as_list_view=True,
    row_selectable='multi',
    ),
        style={'width': '97%', 'float': 'right'},
),
```

Algoritmo 8. Exemplo de layout construído no projeto.

A Figura 10 mostra como a tabela mencionada no algoritmo 8 é apresentada ao usuário, nela é possível ver algumas cervejas cadastradas e algumas características da tabela.

Figura 10. Interface Gráfica que mostra a lista de cervejas adicionadas para otimização.

A partir do entendimento de como o *layout* é gerado e o seu funcionamento, pode-se abordar a segunda parte da *Aplicação web*, o *callback*. Para fazer a interação das informações e realizar o cadastro, é necessário utilizar o *callback*, pois ele é responsável por gerenciar os dados fornecidos pelo usuário e realizar interações através de funções que estão inseridas no *callback*.

Como mostra a Seção 2.4 *Biblioteca Dash*, o *callback* é responsável por gerir as funções que estão disponíveis na tela do sistema. O algoritmo 9 mostra um pouco do funcionamento do cadastro de cervejas na tabela da Figura 10, nesse algoritmo é apresentado a manipulação das informações fornecidas pelo usuário na tabela.

```
1  @app.callback(
2      [    Output("data-table", "data"),
3           Output("data-table", "selected_rows")],
4      [    Input('button_add', 'n_clicks'),
5           Input('button_delete_all', 'n_clicks'),
6           Input('button_delete_items', 'n_clicks')],
7      [    State("list_beer", "value"),
8           State("bulk_beer", "value"),
9           State("data-table", "selected_rows"),
10          State("data-table", "data")]
11     )
12 def update_data(n_clicks_add, n_clicks_delete_all,
       n_clicks_delete_items, beer, bulk, selected_data, data):
13
14     ctx = dash.callback_context
15     if not ctx.triggered:
16         button_id = 'No clicks yet'
17         return create(), []
18     else:
19         button_id = ctx.triggered[0]['prop_id'].split('.')[0]
20
21     L = data.copy()
22
23     if button_id == 'button_add':
24         if all([beer, bulk]):
25             if not isin(L, beer):
26                 L = add(L,beer,bulk)
27
28     elif button_id == 'button_delete_all':
29         L = create()
30
31     elif button_id == 'button_delete_items':
32         L = remove_by_indexes(L, selected_data)
33
34     return L, []
```

Algoritmo 9. Exemplo de callback construído no projeto.

Na linha 1 até a linha 11 é criado o *callback*, dentro dessa função na linha 2 até linha 3 são informados as fontes de saída ao final da aplicação, na linha 2 mostra as informações da tabela e a linha 3 mostra as linhas da tabela selecionadas. Após isso são informadas as variáveis selecionadas pelo usuário da linha 4 até a linha 6, a linha 4 recebe a informação do botão *Adicionar*, a linha 5 recebe a informação do botão *Limpar Lista* e a linha 6 recebe a informação do botão *Remover Item*. Por fim a função recebe os estados dos campos e funções da tabela, da linha 7 até a linha 10, a linha 7 recebe o estado do campo *cerveja*, na linha 8 recebe o estado do campo do *volume*, na linha 9 recebe o estado das linhas selecionadas da tabela e na linha 10 recebe o estado da tabela.

Em seguida é criada a função *update data* da linha 12 até a linha 34, essa função é responsável por atualizar as informações da tabela, dentre as funções fornecidas estão algumas verificações como a do clique dos botões, os valores nulos ou vazios, algumas ações dos botões como adicionar a cerveja e o volume, remover todas as cervejas da lista e remover cervejas selecionados pelo usuário. A linha 12 define as variáveis que a função recebe, que são os cliques dos botões, os campos e as funções da tabela. Após isso na

linha 14 é definido o *ctx* que é um contexto de clique, na linha 15 até a linha 19 é feito a verificação do *ctx* através do *triggered* que é a lista de propriedades alteradas, nesse caso a linha 15 verifica se está vazia, caso esteja na linha 16 o *button id* recebe uma mensagem de que não foi clicado, na linha 17 retorna duas listas vazias, caso o botão já tenha sido clicado na linha 19 o *button id* recebe o contexto do botão, que é a propriedade do clique. Na linha 21 é definido uma lista com as informações da tabela, após isso é verificado os contextos dos cliques na linha 23, na linha 28 e na linha 31. Na linha 23 até a linha 26 executa a ação de adicionar, na linha 28 até a linha 29 executa a ação de limpar a lista e na linha 31 até a linha 32 executa a ação de remover itens selecionados. Por fim na linha 34 retorna a lista com as alterações e uma lista vazia que representa as linhas selecionadas da tabela.

Após compreender o funcionamento das duas partes que compõe a *Aplicação web*, é necessário apresentar o resultado da união das duas partes e entender o funcionamento como uma só ferramenta.

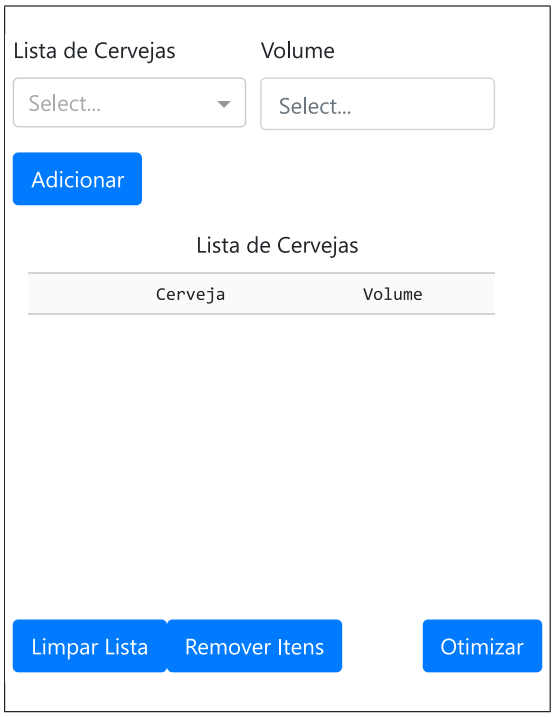

Figura 11. a

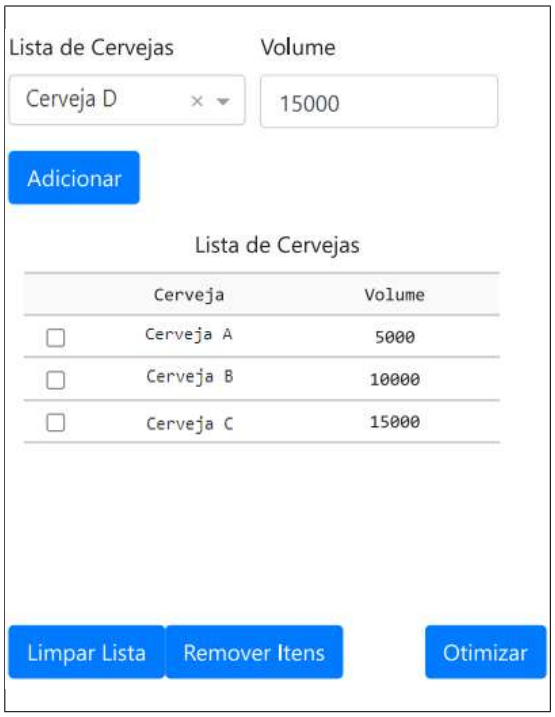

Figura 12. b

A Figura 11 mostra a tela de cadastro das cervejas vazia, sem nenhuma informação selecionada e a Figura 12 mostra a tabela com três cervejas adicionadas, mostrando as opções de manipulação das informações apresentadas no algoritmo 9, como adicionar novas cervejas, remover todos os itens e remover itens selecionados. Com isso, pode-se entender através das Figura 11 e Figuras 12 a representação da união e a interação do *layout* com o *callback*, formando uma única aplicação.

3.4. Teste e Validação

Nesta seção é apresentada uma avaliação do desempenho de otimização para diferentes cenários de volume de produção. O desempenho foi medido pelo tempo de execução

até chegar na melhor solução encontrada, o tipo de solução (ótima ou viável), e o tempo em que será produzido o volume de produção (*makespan*). Para este estudo foi definido um único cenário fabril constituído por: 2 linhas, 18 fermentadores, 11 maturadores, 11 maturadores e fermentadores (tipo flex), 2 centrífugas, 2 filtradores, 2 linhas de dosagem, 2 linhas de recolha e 2 linhas de maturação, sendo que a capacidade de cada tanque foi fixada em $c = 5000\ hl$ (hectolitros). Desse modo, o número de lotes que passam pela linha de produção a partir do volume é dado por $n = c/v$, onde v é o volume que se deseja produzir. Os tempos das etapas de produção de cada lote de cerveja foram definidos da seguinte forma:

1. Arriar: 4 ut (unidades de tempo de produção);
2. Fermentar (1): 16 ut;
3. Fermentar (2): 16 ut;
4. Recolher: 2 ut;
5. CIP recolha: 1 ut;
6. Centrifugação: 4 ut;
7. CIP centrífuga: 1 ut;
8. CIP fermentador: 2 ut;
9. Maturar: 16 ut;
10. Filtragem: 4 ut;
11. CIP linha de maturação: 2 ut;
12. CIP maturador: 2 ut;

Com isso o Tempo Total de Produção de um lote é, em soma 70 ut, mas na realidade 64 ut com certas tarefas de CIP ocorrendo de maneira concomitante com outras tarefas.

Nesta avaliação experimental foram utilizados diferentes volumes de produção, de 5.000 a 35.000 hl. Os experimentos foram executados em um computador Intel Core i7-7700HQ CPU 2.80GHz com 4 núcleos físicos e 16 GB de memória RAM. É importante ressaltar que, como a biblioteca Pyschedule não tem suporte a processamento em múltiplos núcleos, apenas um núcleo foi utilizado no experimento, e a memória RAM utilizada pelo programa não ultrapassou 15 GB no experimento de maior volume de produção. O resultado da avaliação é apresentado na Tabela 1.

#	Volume (hl)	Tempo (seg)	Memória (GB)	Solução	*Makespan (ut)*
1	*5.000	60,38	0,1	ótima	64
2	10.000	135,38	0,5	ótima	68
3	15.000	202,13	0,8	ótima	72
4	20.000	488,19	1,1	ótima	76
5	25.000	1018,53	1,3	ótima	80
6	30.000	1311,02	1,5	ótima	84
7	35.000	7405,95	1,6	ótima	88

Tabela 1. Resultados da avaliação de desempenho da otimização.

Analisando os dados apresentados na Tabela 1 observa-se que, para otimizar o agendamento de tarefas para produção de 5.000 hl (um lote), foram necessários $60,38$ segundos, e foi encontrada uma solução ótima de produção de 64 ut de produção. Observa-se que o tempo para produção dos dois lotes de 5.000 hl (totalizando 10.000 hl) consiste no tempo de execução das tarefas de produção acrescido de 4 ut, em função da concorrência por algum recurso. Para a produção de 30.000 hl, foram necessários $1311,02$ segundos (aproximadamente 22 minutos) para encontrar uma solução ótima de 84 ut. E para a produção de 35.000 hl foram necessários $7405,95$ segundos (aproximadamente duas horas) para encontrar uma solução ótima de 88 ut. Assim, observa-se que o tempo de execução para encontrar uma solução ótima cresce muito rapidamente, mesmo o volume crescendo linearmente. Deve ser observando que os valores apresentados na Tabela 1 são provenientes de apenas uma execução e, como a otimização é um processo não determinístico, pode haver alterações no tempo de execução ao executar a experiência novamente. Vale ressaltar que não foi possível fazer alocação de memória RAM para melhorar o velocidade de processamento dos testes.

No gráfico da Figura 13 é apresentado um gráfico do tempo de execução em função do volume de produção.

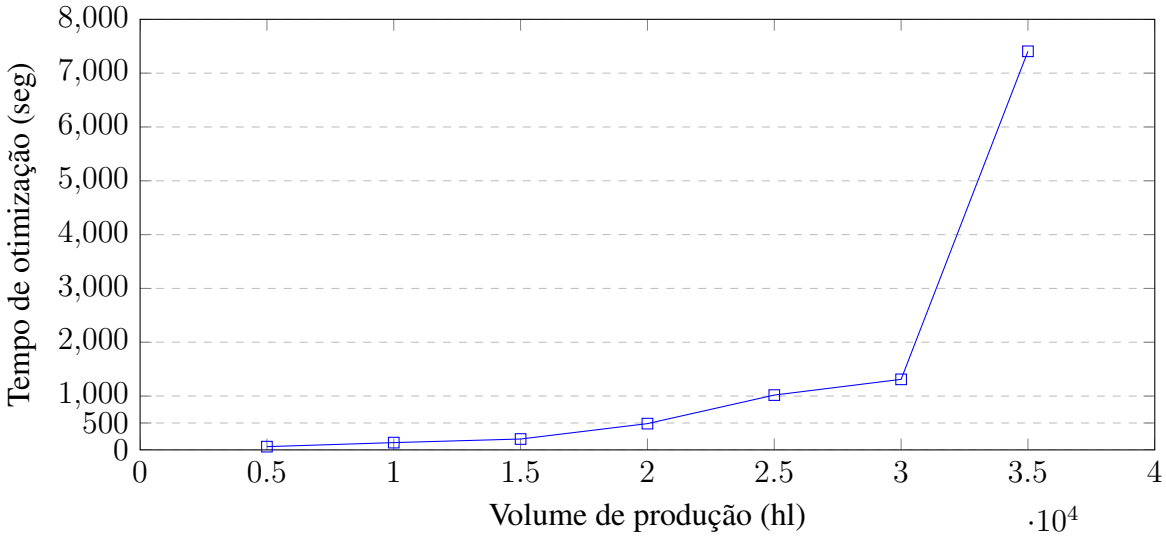

Figura 13. Gráfico do tempo de otimização *vs* o volume de produção.

Por meio do gráfico da Figura 13 é possível observar o aumento do tempo de otimização em função do volume de produção. Embora o volume de produção cresça linearmente, o tempo de execução para otimização cresce em proporções maiores. Por exemplo, em relação ao tempo para otimizar 5.000 hl, os próximos volumes apresentaram $2,25\times$, $3,36\times$, $8,13\times$, $16,9\times$, $21,9\times$ e $123,3\times$ mais tempo para encontrar uma solução ótima. Com isso, também observa-se que à medida que aumenta o volume de produção a busca pela solução ótima torna-se impraticável em termos de tempo de execução, e fica a alternativa de encontrar uma solução viável, de preferência, próxima à solução ótima, que pode ser desconhecida.

Na Figura 14 é apresentado o Diagrama de Gantt do resultado da otimização do agendamento de tarefas para o maior lote da avaliação, 35.000 hl (lotes de 0 a 6). Nesse gráfico, o eixo das abcissas corresponde ao tempo de produção (em ut), o eixo das ordenadas corresponde aos recursos utilizados na produção, as cores representam os lotes (de 0 a 6), e as barras coloridas representam o tempo que um determinado recursos ficou ocupado para atender a produção de determinado lote. Entre os recursos, as siglas *od* e *MB* correspondem a tanques fermentadores e/ou maturadores e bombas de CIP, respectivamente.

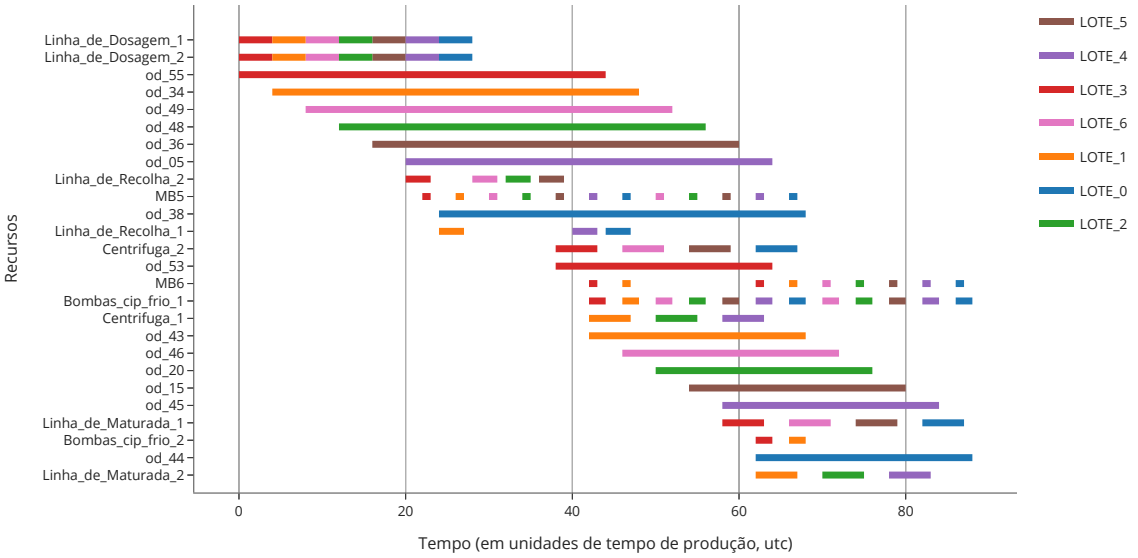

Figura 14. Diagrama de Gantt do agendamento de tarefas para 35.000 hl (lotes de 0 a 6).

Além da avaliação experimental, também foi realizada uma entrevista com um gerente fabril responsável pelo setor de produção de uma fábrica de cervejas da região de Lages, com o intuito de ser realizado um teste prático com o protótipo dentro da cervejaria e dessa forma poder validar sua solução e performance para o problema de agendamento. O especialista indicou que os resultados alcançados para os volumes de produção utilizados nos experimentos são relevantes e representam até 20% da produção total mensal da cervejaria, constituindo resultados promissores para a realização de trabalhos futuros que possam atender a demanda total da fábrica.

4. Conclusão

O escopo do desenvolvimento deste trabalho foi o agendamento de produção em uma fábrica a partir de um volume de produção desejado, considerando que todos os recursos da fábrica estão completamente disponíveis no momento da otimização. Em situações reais, a fábrica encontra-se em funcionamento, as tarefas podem ser interrompidas pela influência de fatores técnicos e humanos, e podem surgir imprevistos, como falta de insumos, entre outros. Nesse sentido, para que a aplicação possa ser utilizada em ambiente real de produção é necessário que a solução atenda um conjunto mais amplo de funcionalidades.

Neste trabalho foi desenvolvido um protótipo de solução computacional para realizar a otimização do agendamento de tarefas do processo de produção de cervejas. Por meio da leitura de material bibliográfico e entrevistas com especialistas do domínio, foi

realizado o levantamento de tarefas, restrições e as variáveis envolvidas nesse processo produtivo. Foram pesquisados artigos científicos, métodos e ferramentas que possam auxiliar no trabalho, e escolhida a biblioteca Pyschedule, que fornece diversas funcionalidades de alto nível úteis para o problema tratado neste trabalho. Com isso, foi realizado o mapeamento das tarefas, restrições e variáveis para o domínio computacional, utilizando a representação da biblioteca Pyschedule, e construída uma aplicação Web para que o usuário possa realizar a otimização do agendamento de tarefas para diferentes tipos de cerveja e volumes de produção. A solução desenvolvida foi avaliada experimentalmente para otimizar o agendamento de tarefas considerando diferentes volumes de produção de um tipo de cerveja. Essa avaliação mostrou que os resultados foram relevantes para volumes de produção de 5.000 hl até 35.000 hl. De acordo com o especialista do domínio os resultados são promissores por mostrar se possível a otimização das principais tarefas de produção de uma fábrica, no entanto precisam ser realizadas mais avaliações com volumes de produção maiores que possam atender um cenário real de produção.

Ao longo do desenvolvimento deste trabalho de conclusão de curso foram identificados problemas de pesquisa que podem contribuir de forma significativa na resolução do problema de pesquisa, e incluem: estudar algoritmos para otimização que considerem um cenário otimizado existente e possibilite o reagendamento de tarefas; pesquisar bibliotecas computacionais para otimização que tenham suporte para uso de múltiplos núcleos do processador durante a otimização e assim chegar em soluções melhores em menor tempo, fazendo uso integral dos recursos computacionais disponíveis; e ampliar as funcionalidades da aplicação Web, para permitir maior customização da fábrica e das etapas de produção de cada cerveja, bem como novas visualizações, que possam auxiliar de forma efetiva no processo de tomada de decisão associados à fabricação de cerveja.

Referências

Baldo, T. A., Santos, M. O., Almada-Lobo, B., e Morabito, R. (2014). An optimization approach for the lot sizing and scheduling problem in the brewery industry. *Computers & Industrial Engineering*, 72:58–71.

Cordeiro, A. (2020). O mapa da cerveja no brasil. Disponível em: `https://revistabeerart.com/news/cervejarias-brasil`. Acesso em: 15-03-2020.

Goldman, R. P., Mark, S. B., e Mark, J. R. (1997). A constraint-based scheduler for batch manufacturing. Disponível em: `https://www.computer.org/csdl/magazine/ex/1997/01/x1049/13rRUyhaIkO`. Acesso em: 20-03-2020.

MAPA (2020). Anuário da cerveja 2019. Ministério da Agricultura Pecuária e Abastecimento. Disponível em: `http://www.cervbrasil.org.br/novo_site/wp-content/uploads/2020/03/anuario-cerveja-WEB.pdf`. Acesso em: 05-06-2020.

Martins, L. F., Pandolfi, M. A. C., e Coimbra, C. C. (2018). Análise dos indicadores do mercado cervejeiro brasileiro. Disponível em: `https://simtec.fatectq.edu.br/index.php/simtec/article/download/261/213/`. Acesso em: 17-04-2020.

MIP, P. (2020). Mixed-integer linear programming. Disponível em: `https://pypi.org/project/mip/`. Acesso em: 10-04-2020.

Morales, S. G. e Ronconi, D. P. (2016). Formulações matemáticas e estratégias de resolução para o problema job shop clássico. *Production*, 26(3):614–625.

Nonner, T. (2019). pyschedule. Disponível em: https://github.com/timnon/pyschedule. Acesso em: 20-03-2020.

Plotly (2017). Dash user guide. Disponível em: https://dash.plotly.com/. Acesso em: 17-04-2020.

Shen, Z. e Smalov, L. (2018). Comparative performance of genetic algorithm, simulated annealing and ant. Disponível em: https://ieeexplore.ieee.org/abstract/document/8638185. Acesso em: 20-03-2019.

Silva, E. L. e Menezes, E. M. (2005). Metodologia da pesquisa e elaboração de dissertação.

Sommerville, I. (2016). *Software Engineering*. Pearson Education Limited.

Sousa, J. M. (2019). Levantamento de requisitos – o ponto de partida do projeto de software. Disponível em: https://blog.cedrotech.com/. Acesso em: 23-09-2020.

Sáenz-Alanís, C. A., D., J. V., e Salazar-Aguila, M. A. (2016). A parallel machine batch scheduling problem in a brewing company. Disponível em: https://link.springer.com/article/10.1007/s00170-016-8477-8. Acesso em: 21-03-2020.

Wazlawick, R. S. (2013). *Engenharia de Software: Conceitos e Práticas*. Elsevier Editora Ltda.

Zheng, S., Zheng, X., e Wang, C. (2016). The optimization of beer recipe based on an improved ant colony optimizatio*. Disponível em: https://ieeexplore.ieee.org/abstract/document/7515881. Acesso em: 20-03-2020.