Desafio do Bootcamp de Otimização ENACOM 2022

Fabricio Schiavon Kolberg

Agosto 2022

Apresentação Pessoal

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).
- Doutor em Ciência da Computação na Universidade Federal do Paraná (2016-2021).

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).
- Doutor em Ciência da Computação na Universidade Federal do Paraná (2016-2021).

Experiência acadêmica:

- Teoria da computação.

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).
- Doutor em Ciência da Computação na Universidade Federal do Paraná (2016-2021).

Experiência acadêmica:

- Teoria da computação.
- Teoria dos grafos.

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).
- Doutor em Ciência da Computação na Universidade Federal do Paraná (2016-2021).

Experiência acadêmica:

- Teoria da computação.
- Teoria dos grafos.
- Métodos formais.

Apresentação Pessoal

Educação:

- Bacharel em Ciência da Computação na Universidade Federal do Paraná (2008-2013).
- Mestrado em Informática na Universidade Federal do Paraná (2014-2016).
- Doutor em Ciência da Computação na Universidade Federal do Paraná (2016-2021).

Experiência acadêmica:

- Teoria da computação.
- Teoria dos grafos.
- Métodos formais.

Experiência profissional:

- Estágio: LABMET-UFPR (number crunching com Fortran, análise de dados atmosféricos) (2013-2014).

Introdução ao Problema

Introdução ao Problema

Lista de potenciais investimentos:

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Orçamento de R$ 1.000.000.

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Orçamento de R$ 1.000.000.

Restrições adicionais:

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Orçamento de R$ 1.000.000.

Restrições adicionais:

- Se a opção 1 for selecionada, a opção 5 *não* pode ser.

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Orçamento de R$ 1.000.000.

Restrições adicionais:

- Se a opção 1 for selecionada, a opção 5 *não* pode ser.
- Se a opção 2 for selecionada, a opção 4 *também* deve ser.

Introdução ao Problema

Lista de potenciais investimentos:

Opção	Descrição	Custo (R$)	Retorno (R$)
1	Ampliação da capacidade do armazém ZDP em 5%	470.000	410.000
2	Ampliação da capacidade do armazém MGL em 7%	400.000	330.000
3	Compra de empilhadeira	170.000	140.000
4	Projeto de P&D I	270.000	250.000
5	Projeto de P&D II	340.000	320.000
6	Aquisição de novos equipamentos	230.000	320.000
7	Capacitação de funcionários	50.000	90.000
8	Ampliação da estrutura de carga rodoviária	440.000	190.000

Orçamento de R$ 1.000.000.
Restrições adicionais:

- Se a opção 1 for selecionada, a opção 5 *não* pode ser.
- Se a opção 2 for selecionada, a opção 4 *também* deve ser.

Desafio: maximizar o retorno de investimento sem quebrar o orçamento máximo, e respeitando as restrições adicionais.

Solução do Problema

- Ideia: resolver o problema usando um programa em *Python*.

Solução do Problema

- Ideia: resolver o problema usando um programa em *Python*.
- A empresa hipotética pode querer a solução de um problema similar com parâmetros diferentes no futuro.

Solução do Problema

- Ideia: resolver o problema usando um programa em *Python*.
- A empresa hipotética pode querer a solução de um problema similar com parâmetros diferentes no futuro.
- Vale a pena, então, *generalizar* o problema.

Versão Generalizada do Problema

Temos os seguintes dados:

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Sejam $v_1, v_2, ..., v_n$ as *variáveis de decisão* do problema, onde, para todo $1 \leq i \leq n$, $v_i = 1$ se o investimento I_i for escolhido, e $v_i = 0$ caso contrário.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Sejam $v_1, v_2, ..., v_n$ as *variáveis de decisão* do problema, onde, para todo $1 \leq i \leq n$, $v_i = 1$ se o investimento I_i for escolhido, e $v_i = 0$ caso contrário.
Restrições adicionais:

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Sejam $v_1, v_2, ..., v_n$ as *variáveis de decisão* do problema, onde, para todo $1 \leq i \leq n$, $v_i = 1$ se o investimento I_i for escolhido, e $v_i = 0$ caso contrário.

Restrições adicionais:

Restrições de conflito Se os investimentos I_a e I_b conflitam, v_a e v_b não podem ambos ser 1.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Sejam $v_1, v_2, ..., v_n$ as *variáveis de decisão* do problema, onde, para todo $1 \leq i \leq n$, $v_i = 1$ se o investimento I_i for escolhido, e $v_i = 0$ caso contrário.
Restrições adicionais:

Restrições de conflito Se os investimentos I_a e I_b conflitam, v_a e v_b não podem ambos ser 1.

Restrições de dependência Se o investimento I_a depende de I_b, então v_a só pode ser 1 se v_b também for.

Versão Generalizada do Problema

Temos os seguintes dados:

- n potenciais investimentos $I_1, I_2, ..., I_n$.
- Custos $c_1, c_2, ..., c_n$.
- Retornos $r_1, r_2, ..., r_n$.
- Orçamento máximo disponível M.

Sejam $v_1, v_2, ..., v_n$ as *variáveis de decisão* do problema, onde, para todo $1 \leq i \leq n$, $v_i = 1$ se o investimento I_i for escolhido, e $v_i = 0$ caso contrário.
Restrições adicionais:

Restrições de conflito Se os investimentos I_a e I_b conflitam, v_a e v_b não podem ambos ser 1.

Restrições de dependência Se o investimento I_a depende de I_b, então v_a só pode ser 1 se v_b também for.

Modelagem Matemática

Função objetivo:

$$\max \sum_{i=1}^{n} v_i r_i$$

Modelagem Matemática

Função objetivo:

$$\max \sum_{i=1}^{n} v_i r_i$$

Restrição orçamentária:

$$\sum_{i=1}^{n} v_i c_i \leq M$$

Modelagem Matemática

Função objetivo:

$$\max \sum_{i=1}^{n} v_i r_i$$

Restrição orçamentária:

$$\sum_{i=1}^{n} v_i c_i \leq M$$

Restrições de conflito, para todo $1 \leq a, b \leq n$ tal que I_a, I_b conflitam:

$$v_a + v_b \leq 1$$

Modelagem Matemática

Função objetivo:

$$\max \sum_{i=1}^{n} v_i r_i$$

Restrição orçamentária:

$$\sum_{i=1}^{n} v_i c_i \leq M$$

Restrições de conflito, para todo $1 \leq a, b \leq n$ tal que I_a, I_b conflitam:

$$v_a + v_b \leq 1$$

Restrições de dependência, para todo $1 \leq a, b \leq n$ tal que I_a depende de I_b:

$$v_b + (1 - v_a) \geq 1$$

Domínio das variáveis de decisão:

$$\forall 1 \leq i \leq n, v_i \in \{0, 1\}.$$

Modelagem Matemática

Para o problema do desafio:

Modelagem Matemática

Para o problema do desafio:

- $n = 8$

Modelagem Matemática

Para o problema do desafio:

- $n = 8$
- $c_1 = 470000, c_2 = 400000, c_3 = 170000, c_4 = 270000, c_5 = 340000, c_6 = 230000, c_7 = 50000, c_8 = 440000$

Modelagem Matemática

Para o problema do desafio:

- $n = 8$
- $c_1 = 470000, c_2 = 400000, c_3 = 170000, c_4 = 270000, c_5 = 340000, c_6 = 230000, c_7 = 50000, c_8 = 440000$
- $r_1 = 410000, r_2 = 330000, r_3 = 140000, r_4 = 250000, r_5 = 320000, r_6 = 320000, r_7 = 90000, r_8 = 190000$

Modelagem Matemática

Para o problema do desafio:

- $n = 8$
- $c_1 = 470000, c_2 = 400000, c_3 = 170000, c_4 = 270000, c_5 = 340000, c_6 = 230000, c_7 = 50000, c_8 = 440000$
- $r_1 = 410000, r_2 = 330000, r_3 = 140000, r_4 = 250000, r_5 = 320000, r_6 = 320000, r_7 = 90000, r_8 = 190000$
- $M = 1000000$

Modelagem Matemática

Para o problema do desafio:

- $n = 8$
- $c_1 = 470000, c_2 = 400000, c_3 = 170000, c_4 = 270000, c_5 = 340000, c_6 = 230000, c_7 = 50000, c_8 = 440000$
- $r_1 = 410000, r_2 = 330000, r_3 = 140000, r_4 = 250000, r_5 = 320000, r_6 = 320000, r_7 = 90000, r_8 = 190000$
- $M = 1000000$
- $v_1 + v_5 \leq 1$

Modelagem Matemática

Para o problema do desafio:

- $n = 8$
- $c_1 = 470000, c_2 = 400000, c_3 = 170000, c_4 = 270000, c_5 = 340000, c_6 = 230000, c_7 = 50000, c_8 = 440000$
- $r_1 = 410000, r_2 = 330000, r_3 = 140000, r_4 = 250000, r_5 = 320000, r_6 = 320000, r_7 = 90000, r_8 = 190000$
- $M = 1000000$
- $v_1 + v_5 \leq 1$
- $v_4 + (1 - v_2) \geq 1$

Criação do Programa

- Utilizando o pacote *science-optimization* do Python.

Criação do Programa

- Utilizando o pacote *science-optimization* do Python.
- Representação de problemas de otimização do pacote:

Criação do Programa

- Utilizando o pacote *science-optimization* do Python.
- Representação de problemas de otimização do pacote:
  $\min f(x)$
  Sujeito a:
  $g(x) \leq 0$
  $h(x) = 0$

Criação do Programa

Adaptando as expressões do problema para o *science-optimization*.

Criação do Programa

Adaptando as expressões do problema para o *science-optimization*.
Função objetivo:

$$\min \sum_{i=1}^{n} v_i(-r_i)$$

Criação do Programa

Adaptando as expressões do problema para o *science-optimization*.
Função objetivo:

$$\min \sum_{i=1}^{n} v_i(-r_i)$$

Restrição de orçamento:

$$\sum_{i=1}^{n} v_i c_i \leq M \Rightarrow \sum_{i=1}^{n} v_i c_i - M \leq 0.$$

Criação do Programa

Adaptando as expressões do problema para o *science-optimization*.
Função objetivo:

$$\min \sum_{i=1}^{n} v_i(-r_i)$$

Restrição de orçamento:

$$\sum_{i=1}^{n} v_i c_i \leq M \Rightarrow \sum_{i=1}^{n} v_i c_i - M \leq 0.$$

Conflitos:

$$v_a + v_b \leq 1 \Rightarrow v_a + v_b - 1 \leq 0$$

Criação do Programa

Adaptando as expressões do problema para o *science-optimization*.
Função objetivo:

$$\min \sum_{i=1}^{n} v_i(-r_i)$$

Restrição de orçamento:

$$\sum_{i=1}^{n} v_i c_i \leq M \Rightarrow \sum_{i=1}^{n} v_i c_i - M \leq 0.$$

Conflitos:

$$v_a + v_b \leq 1 \Rightarrow v_a + v_b - 1 \leq 0$$

Dependências:

$$v_b + (1 - v_a) \geq 1 \Rightarrow -v_b - 1 + v_a \leq -1 \Rightarrow -v_b - 1 + v_a + 1 \leq 0 \Rightarrow$$
$$-v_b + v_a \leq 0$$

Criação do Programa

Estruturas de dados básicas para *entrada*:

Criação do Programa

Estruturas de dados básicas para *entrada*:

Custos Lista de valores numéricos $[c_1, ..., c_n]$.

Criação do Programa

Estruturas de dados básicas para *entrada*:

Custos Lista de valores numéricos $[c_1, ..., c_n]$.

Retornos Lista de valores numéricos $[r_1, ..., r_n]$.

Criação do Programa

Estruturas de dados básicas para *entrada*:

Custos Lista de valores numéricos $[c_1, ..., c_n]$.

Retornos Lista de valores numéricos $[r_1, ..., r_n]$.

Orçamento Valor numérico.

Criação do Programa

Estruturas de dados básicas para *entrada*:

Custos Lista de valores numéricos $[c_1, ..., c_n]$.

Retornos Lista de valores numéricos $[r_1, ..., r_n]$.

Orçamento Valor numérico.

Conflitos Lista de pares de índices (onde índice i representa o investimento $i + 1$) $[[a_1, b_1], ..., [a_k, b_k]]$.

Criação do Programa

Estruturas de dados básicas para *entrada*:

Custos Lista de valores numéricos $[c_1, ..., c_n]$.

Retornos Lista de valores numéricos $[r_1, ..., r_n]$.

Orçamento Valor numérico.

Conflitos Lista de pares de índices (onde índice i representa o investimento $i + 1$) $[[a_1, b_1], ..., [a_k, b_k]]$.

Dependências Lista de pares de índices (onde índice i representa o investimento $i + 1$) $[[a_1, b_1], ..., [a_l, b_l]]$.

Criação do Programa

Código de entrada:

Criação do Programa

Código de entrada:

```
# INPUTS (prices, returns, budget, conflict, dependencies)
# All separated by a line break
# (it ties into the instance generator I made in Bash)
instance_size = int(input()) # investment options number
prices = []
returns = []
names = []
budget = int(input())
for i in range(0,instance_size):
    prices.append(int(input()))
for i in range(0,instance_size):
    returns.append(int(input()))
# names array: only important for the console output
for i in range(0,instance_size):
    names.append(input())
# conflict array: for every item [a,b] in the array, a and b cannot be
# chosen at the same time.
conflict_size = int(input())
conflicts = []
for i in range(0,conflict_size):
    arrayzinha = [-1,-1]
    arrayzinha[0] = int(input())
    arrayzinha[1] = int(input())
    conflicts.append(arrayzinha)
# dependency array: for every item [a,b] in the array, item a can only be
# picked if b also is.
depends_size = int(input())
dependencies = []
for i in range(0,depends_size):
    arrayzinha = [-1,-1]
    arrayzinha[0] = int(input())
    arrayzinha[1] = int(input())
    dependencies.append(arrayzinha)
```

Criação do Programa

Código de criação do objeto do problema:

Criação do Programa

Código de criação do objeto do problema:

```
# Problem class
class InvestmentProblem(BuilderOptimizationProblem):
    # First reading of the problem info:
    # prices and returns are converted to numpy because we will
    # need to use them as coefficients in the restrictions and
    # objective function.
    def __init__(self, prices: List[int], returns: List[int],
                 budget: int, conflicts: List[List[int]],
                 dependencies: List[List[int]]):
        self.__varnum = len(prices)
        self.__prices = numpy.array([x for x in prices])
        self.__returns = numpy.array([x for x in returns])
        self.__budget = budget
        self.__conflicts = conflicts
        self.__dependencies = dependencies
```

Criação do Programa

Criação das variáveis:

Criação do Programa

Criação das variáveis:

- Temos n (__varnum) variáveis.

Criação do Programa

Criação das variáveis:

- Temos n (__varnum) variáveis.
- Mínimo 0, máximo 1.

Criação do Programa

Criação das variáveis:

- Temos n (__varnum) variáveis.
- Mínimo 0, máximo 1.
- Variáveis *discretas*.

Criação do Programa

Criação das variáveis:

- Temos n (__varnum) variáveis.
- Mínimo 0, máximo 1.
- Variáveis *discretas*.

```
# building variables: pretty much the same thing we saw in the
# second class.
def build_variables(self):
    lower = numpy.zeros((self.__varnum, 1))
    upper = numpy.ones((self.__varnum, 1))
    # all discrete, naturally
    types = ['d']*self.__varnum
    variables = Variable(lower, upper, types)
    return variables
```

Criação do Programa

Expressões de *LinearFunction()* em *science-optimization*:

Criação do Programa

Expressões de *LinearFunction()* em *science-optimization*:

$$a_1 x_1 + \ldots + a_n x_n + d$$

Criação do Programa

Expressões de *LinearFunction()* em *science-optimization*:

$$a_1 x_1 + \ldots + a_n x_n + d$$

Sintaxe da função de criação do objeto:

Criação do Programa

Expressões de *LinearFunction()* em *science-optimization*:

$$a_1 x_1 + ... + a_n x_n + d$$

Sintaxe da função de criação do objeto:

$$LinearFunction(c = [[a_1], ..., [a_n]], d = d)$$

Criação do Programa

Função objetivo:

```
def build_objectives(self):
    fun = LinearFunction(c=-self.__returns.reshape(-1,1),d=0)
    funs = FunctionsComposite()
    funs.add(fun)
    obj = Objective(objective=funs)
    return obj
```

Criação do Programa

Função objetivo:

```
def build_objectives(self):
    fun = LinearFunction(c=-self.__returns.reshape(-1,1),d=0)
    funs = FunctionsComposite()
    funs.add(fun)
    obj = Objective(objective=funs)
    return obj
```

Restrição de orçamento:

```
# building constraints: making sure to add the conflicts and
# dependencies
def build_constraints(self):
    cons_set = FunctionsComposite()
    # first: budget constraint. Pretty much the same as the knapsack
    # example we saw in class
    budgetary = LinearFunction(c=self.__prices.reshape(-1,1), d=-self.__budget)
    cons_set.add(budgetary)
```

Criação do Programa

Restrições de conflito e dependência:

```
# "base" for the coefficients in the rest of the constraints:
# set all coefficients to 0 at first, and then change only the
# coefficients of the relevant variables.
base = numpy.zeros((self.__varnum, 1))
# conflict constraints.
# basically, if a and b conflict, then
# a+b <= 1, that is, a+b-1 <= 0
for item in self.__conflicts:
    a = item[0]
    b = item[1]
    base[a] = 1
    base[b] = 1
    newcons = LinearFunction(c=base.reshape(-1,1),d=-1)
    cons_set.add(newcons)
    base = numpy.zeros((self.__varnum, 1))

# dependency constraints.
# if a depends on b, then
# -b+a <= 0
for item in self.__dependencies:
    a = item[0]
    b = item[1]
    base[a] = 1
    base[b] = -1
    newcons = LinearFunction(c=base.reshape(-1,1),d=0)
    cons_set.add(newcons)
    base = numpy.zeros((self.__varnum, 1))

# ALRIGHT, WRAP IT UP, IT'S DONE.
constraints = Constraint(ineq_cons=cons_set)
return constraints
```

Criação do Programa

Execução do problema e impressão do resultado:

```
# Computation of the tests:
problem = InvestmentProblem(prices,returns,budget,conflicts,dependencies)
opt = OptimizationProblem(builder=problem)
optimizer = Optimizer(opt_problem=opt,algorithm=Glop())
results = optimizer.optimize()
whichones = results.x.ravel()
totalcost = 0
print("Investmentos escolhidos:")
for i in range(0,len(prices)):
    if whichones[i] == 1.:
        print(names[i])
        totalcost += prices[i]
print("Retorno total: ",-int(results.fx[0]))
print("Custo total: ",totalcost)
```

Resultados

Primeiramente, os resultados para o problema do desafio:

Resultados

Primeiramente, os resultados para o problema do desafio:

```
Investmentos escolhidos:
Ampliar armazem MGL em 7%
Projeto de P&D I
Aquisição de novos equipamentos
Capacitação de funcionários
Retorno total:  990000
Custo total:  950000
```

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.
Para cada número de investimentos entre 1 e 4078, dez instâncias foram geradas, e o tempo de execução médio do programa foi medido para cada uma.

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.
Para cada número de investimentos entre 1 e 4078, dez instâncias foram geradas, e o tempo de execução médio do programa foi medido para cada uma.
Máquina usada:

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.
Para cada número de investimentos entre 1 e 4078, dez instâncias foram geradas, e o tempo de execução médio do programa foi medido para cada uma.
Máquina usada:

- Processador *AMD FX-8350 Eight-core* (4 GHz).

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.
Para cada número de investimentos entre 1 e 4078, dez instâncias foram geradas, e o tempo de execução médio do programa foi medido para cada uma.
Máquina usada:

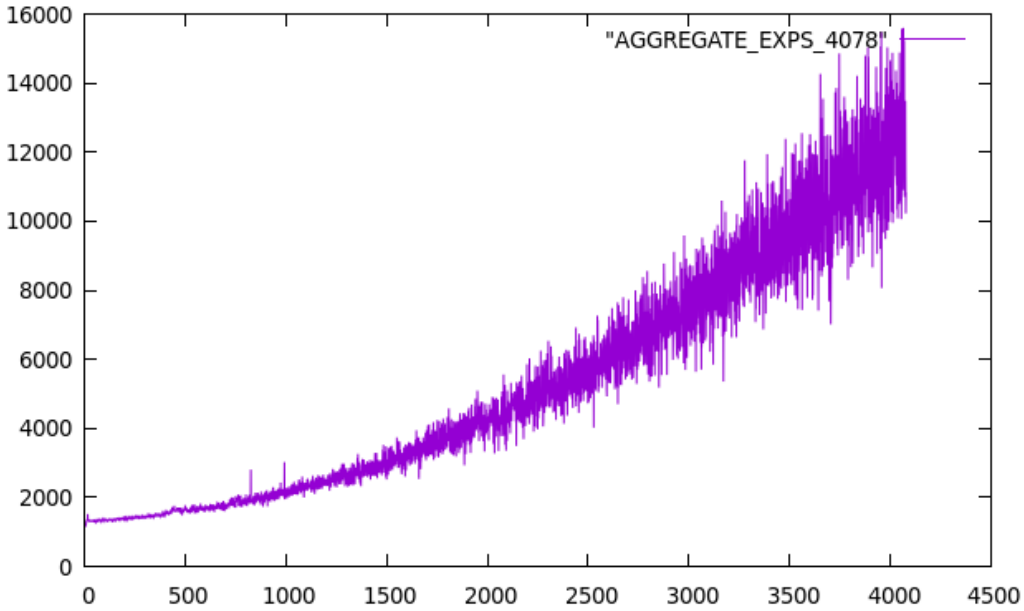
- Processador *AMD FX-8350 Eight-core* (4 GHz).
- 8 GB de memória RAM *DDR 3*.

Resultados

Para estudar o desempenho do programa mais a fundo, foi criado um *gerador de instâncias*.
Para cada número de investimentos entre 1 e 4078, dez instâncias foram geradas, e o tempo de execução médio do programa foi medido para cada uma.
Máquina usada:

- Processador *AMD FX-8350 Eight-core* (4 GHz).
- 8 GB de memória RAM *DDR 3*.
- Rodando uma máquina virtual de *Linux Mint* usando *OracleVM VirtualBox* em *Windows 7* de 64 bits.

Resultados



- Eixo x : tempo em milisegundos.
- Eixo y : número de opções.

Resultados

- O aumento do tempo levado em função do tamanho da instância é acima de linear (como esperado, pois o problema é exponencial no pior caso).

Resultados

- O aumento do tempo levado em função do tamanho da instância é acima de linear (como esperado, pois o problema é exponencial no pior caso).
- Além disso, quanto maiores os tamanhos de instância, mais "espalhados" os tempos de execução ficaram.

Resultados

- O aumento do tempo levado em função do tamanho da instância é acima de linear (como esperado, pois o problema é exponencial no pior caso).
- Além disso, quanto maiores os tamanhos de instância, mais "espalhados" os tempos de execução ficaram.
  - Hipótese 1: quanto maior o número de opções, maior o impacto do número de restrições sobre o tempo de execução (isso pode ser testado fixando o número de restrições).

Resultados

- O aumento do tempo levado em função do tamanho da instância é acima de linear (como esperado, pois o problema é exponencial no pior caso).
- Além disso, quanto maiores os tamanhos de instância, mais "espalhados" os tempos de execução ficaram.
  - Hipótese 1: quanto maior o número de opções, maior o impacto do número de restrições sobre o tempo de execução (isso pode ser testado fixando o número de restrições).
  - Hipótese 2: o número de testes por tamanho de instância (10) pode ter sido insuficiente para determinar uma média consistente para os tempos de execução (isso pode ser testado aumentando o número de instâncias para cada tamanho).

Resultados

- O aumento do tempo levado em função do tamanho da instância é acima de linear (como esperado, pois o problema é exponencial no pior caso).
- Além disso, quanto maiores os tamanhos de instância, mais "espalhados" os tempos de execução ficaram.
  - Hipótese 1: quanto maior o número de opções, maior o impacto do número de restrições sobre o tempo de execução (isso pode ser testado fixando o número de restrições).
  - Hipótese 2: o número de testes por tamanho de instância (10) pode ter sido insuficiente para determinar uma média consistente para os tempos de execução (isso pode ser testado aumentando o número de instâncias para cada tamanho).
  - Hipótese 3: devido à forma como preços e retornos foram atribuídos de forma aleatória a cada opção no gerador de instâncias, existe a possibilidade de que certas instâncias tenham contido opções "super-poderosas" de baixo custo e alto retorno, que grandemente simplificaram o trabalho do algoritmo de otimização (isso pode ser testado calculando os retornos das opções em função de seus custos).

Resultados

- De qualquer forma, mesmo nos casos com mais de 4000 opções, o programa demorou menos de 15 segundos, em média, para encontrar a solução ótima.

Resultados

- De qualquer forma, mesmo nos casos com mais de 4000 opções, o programa demorou menos de 15 segundos, em média, para encontrar a solução ótima.
- Isso se deve, principalmente, às otimizações existentes no pacote *science-optimization* e no algoritmo *Glop*.