São José dos Campos - SP
LV SBPO
LV Simpósio Brasileiro de Pesquisa Operacional
São José dos Campos, SP - 6 a 9 de novembro de 2023

`QUBO.jl`: O ecossistema em Julia para Otimização Quântica

Pedro Maciel Xavier, Pedro Ripper, Tiago Andrade, Joaquim Dias Garcia
PSR
Rio de Janeiro 22250-040, Brazil
{pedroxavier,pedroripper,tiago.andrade,joaquim}@psr-inc.com

RESUMO

Apresentamos `QUBO.jl`, um ecossistema Julia para trabalhar com QUBO (*Quadratic Unconstrained Binary Optimization*). Nossa ferramenta foi concebida para se tornar a ponte entre a Pesquisa Operacional e a Otimização Quântica, fornecendo métodos para converter problemas gerais modelados na biblioteca `JuMP` para otimização em reformulações no formato QUBO, que são executáveis em computadores quânticos e outras arquiteturas emergentes, como *annealers* quânticos, máquinas de Ising e máquinas de bifurcação simulada. Além de reformular os modelos de otimização, nosso software é capaz de interagir com os dispositivos mencionados, exigindo nenhum conhecimento prévio específico de sua operação, permitindo que os usuários facilmente apresentem seus problemas e analisem os resultados obtidos.

PALAVRAS CHAVE. QUBO, Julia, Modelagem Matemática, Computação Quântica

TÓPICOS: OC – Otimização Combinatória, PM – Programação Matemática, OA – Outras aplicações em PO

ABSTRACT

We introduce `QUBO.jl`, a Julia ecosystem for working with QUBO (Quadratic Unconstrained Binary Optimization). Our tool was envisioned to become the bridge between Operations Research and Quantum Optimization, providing methods to convert general `JuMP` problems into the QUBO format, which are executable in quantum computers and other emerging architectures, such as quantum annealers, coherent ising machines and simulated bifurcation machines. Along with reformulating optimization models, our software is able to interface with the aforementioned devices, without any prior knowledge on how to operate on them, allowing users to easily submit problems and analyse the subsequent results.

KEYWORDS. QUBO, Julia, Mathematical Modeling, Quantum Computing

TOPICS: CO - Combinatorial Optimization, MP – Mathematical Programming, OA – Othe applications in OR

1. Introdução

Sucessivos avanços no desenvolvimento de algoritmos e hardware no campo da Computação Quântica têm despertado crescente interesse tanto na academia como na indústria. Esse interesse é fomentado pelas expectativas de ganho computacional que a tecnologia poderá oferecer em diversas tarefas, dentre elas a solução de problemas de otimização combinatória.

Esse trabalho apresenta o `QUBO.jl`, um software de Programação Matemática para se trabalhar com modelos QUBO (*Quadratic Unconstrained Binary Optimization*) em computadores quânticos e alguns outros hardwares de vanguarda. Dentre tais plataformas, destacam-se os "*Quantum Annealers*", os "*Digital Annealers*", as "*Simulated Bifurcation Machines*", e também implementações de "*Simulated Annealing*" e "*Parallel Tempering*" para computadores digitais convencionais. A ferramenta tem como principais funcionalidades 1. a tradução de problemas de otimização para o formato QUBO; 2. a comunicação com dispositivos compatíveis com a reformulação gerada; 3. a recuperação, formatação e análise dos resultados obtidos. Através da linguagem de modelagem algébrica `JuMP` [Dunning et al., 2017], o usuário é capaz de percorrer, de maneira fluida, todo o roteiro de trabalho usual de otimização sem que seja necessário profundo conhecimento de mecânica quântica.

2. O ecossistema `QUBO.jl`

O `QUBO.jl` é um pacote da linguagem Julia capaz de englobar todo o fluxo de trabalho para lidar com modelos QUBO. Com ele, pode-se converter, de forma automática, problemas gerais de Otimização modelados no `JuMP` para o formato em destaque. Considerando o significado do acrônimo QUBO, o modelo final, depois da tradução, deve ter somente variáveis binárias, ser irrestrito e representado por um polinômio de grau no máximo 2.

Dessa forma, durante a reformulação realizada pelo `QUBO.jl`, três etapas são necessárias. Primeiramente, é preciso codificar variáveis não-binárias usando somente binárias.

Posteriormente, a remoção de restrições de um problema pode ser realizada adicionando termos de penalidade à função objetivo. Finalmente, para manter o polinômio no máximo quadrático, temos que *quadratizar* o modelo resultante.

As instâncias de QUBO geradas podem ser posteriormente enviadas para computadores quânticos e outras arquiteturas, previamente mencionados, com os quais o usuário pode se comunicar como se fossem solvers tradicionais compatíveis com `JuMP`. Isso é possível uma vez que, com o `QUBO.jl`, pode-se encapsular a comunicação com diferentes hardwares compatíveis com QUBO, criando uma camada de abstração. Depois que as soluções para o problema são amostradas, o `QUBO.jl` oferece ferramentas de análise de resultados e métodos de visualização.

O nosso pacote se destaca em relação aos seus concorrentes em diferentes aspectos. Primeiramente, usando o despacho múltiplo da linguagem Julia, o `QUBO.jl` e seus submódulos podem ser estendidos, possibilitando comunicação com novos hardwares e definição de novos métodos de reformulação, por exemplo. Além disso, foram implementados mais métodos de codificação de variáveis e mapeamento de restrições, como mostrado na Tabela 1, o que permite o usuário trabalhar com uma maior gama de problemas de otimização.

Ademais, em um *benchmark*, o `QUBO.jl` demonstrou melhor performance de tempo para reformular problemas para o formato QUBO, como demonstrado na Figura 1, onde foram usados os problemas do Caixeiro-viajante e da Partição de Números para comparação. O problema do Caixeiro-viajante demanda a representação de um grande número de restrições, enquanto que o de Partição de Números já pode ser entendido como um QUBO em sua formulação original. Assim, a escolha dos casos de teste revela que o `QUBO.jl` demonstra uma vantagem de escalabilidade

em função do número de variáveis e restrições, da mesma forma que possui um fator constante de modelagem reduzido.

Tabela 1: Codificação e Mapeamento Automáticos de Variáveis e Restrições

	`QUBO.jl`	`PyQUBO` [3]	`qubovert` [4]	`Amplify` [5]
Codificação Automática de Variáveis				
Binary [1]	■	■	■	■
Unary [1]	■	■	■	■
One-Hot [1]	■	■		
Domain-Wall [2]	■	■		
Mapeamento Automático de Restrições				
$\vec{a}'\vec{x} \leq b$	■		■	■
$\vec{a}'\vec{x} = b$	■		■	■
$\vec{x}'Q\,\vec{x} + \vec{a}'\vec{x} \leq b$	■			■
$\vec{x}'Q\,\vec{x} + \vec{a}'\vec{x} = b$	■			■

[1] [Tamura et al., 2021]; [2] [Chancellor, 2019]; [3] [Zaman et al., 2021] [4] [Iosue, 2022] [5] [Matsuda, 2022]

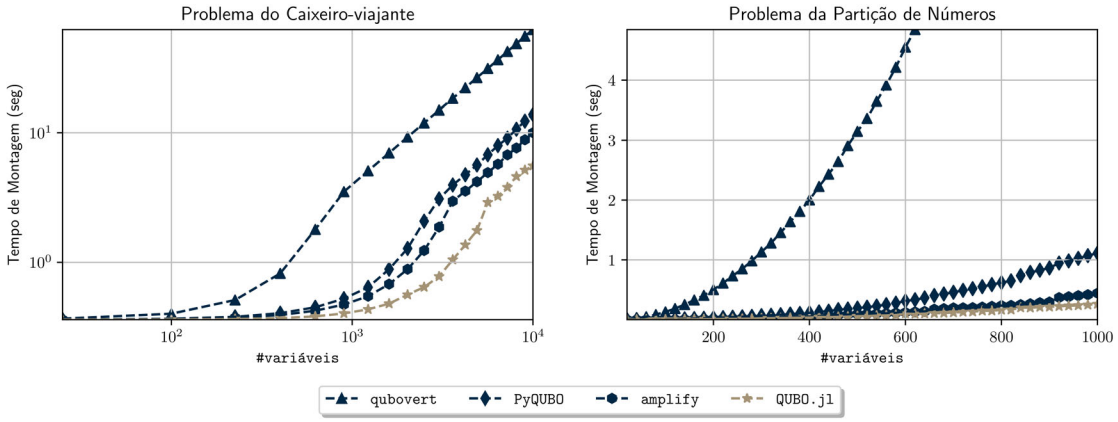

Figura 1: Problema do Caixeiro-viajante com $\sqrt{\texttt{\#variáveis}}$ cidades e Problema da Partição de Números com $[\texttt{\#variáveis}] = \{1, \ldots, n\}$

A convenção adotada para representar o problema do Caixeiro-viajante nos casos do *benchmark* é a da formulação por permutação de cidades. Nesse formato, dadas n cidades, serão utilizadas n^2 variáveis binárias $x_{i,k}$. Quando $x_{i,k} = 1$, temos que i-ésima cidade fora alocada na k-ésima posição da *tour*, que nada mais é que a sequência de cidades percorridas.

$$\begin{array}{lll} \min\limits_{\mathbf{x}} & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} D_{i,j}\, x_{i,k}\, x_{j,(k \bmod n)+1} & \\ \text{sujeito a} & \sum_{k=1}^{n} x_{i,k} = 1 & \forall i \\ & \sum_{i=1}^{n} x_{i,k} = 1 & \forall k \\ & x_{i,k} \in \{0,1\} & \forall i,k \end{array} \tag{1}$$

onde $D_{i,j}$ é a distância entre as cidades i e j. Para a realização dos experimentos, foram produzidos programas específicos de modelagem para cada plataforma. A formulação matemática de cada

problema, contudo, foi preservada independentemente da interface avaliada. Foi dada prioridade à adaptação de trechos de código pertinentes que já constassem na documentação dos projetos, em detrimento de novas implementações.

O Código 1 mostra como o `QUBO.jl` foi utilizado para modelar o problema do Caixeiro-viajante, exemplo que ressalta a praticidade da ferramenta. Comparando com a Formulação (1), podemos ver o quanto o código em `JuMP` se aproxima da linguagem matemática usada para a representação do problema.

Código 1: Código do `QUBO.jl` para reformular o problema do Caixeiro-viajante para um QUBO

```
using QUBO

function caixeiro_viajante(n::Integer, D::Matrix)
    modelo = Model(ToQUBO.Optimizer)

    @variable(modelo, x[1:n, 1:n], Bin)

    @objective(
        modelo,
        Min,
        sum(D[i,j] * x[i,k] * x[j, k%n+1] for i = 1:n, j = 1:n, k = 1:n)
    )
    @constraint(modelo, [i in 1:n], sum(x[i,:]) == 1)
    @constraint(modelo, [k in 1:n], sum(x[:,k]) == 1)

    # Compilation
    optimize!(modelo)
    Q, a, b = QUBO.qubo(modelo, Dict)
end
```

Sendo o ecossistema `QUBO.jl` uma solução para integrar o campo da Pesquisa Operacional com a Computação Quântica, em versões futuras é esperado que o software suporte uma maior gama de problemas de otimização e disponibilize conexão com um maior número de computadores quânticos. O primeiro objetivo será alcançado com melhorias no arcabouço de codificação de variáveis e mapeamento de restrições. Já o segundo poderá ser cumprido com adaptações na interface que considerem as particularidades de cada arquitetura.

Referências

Chancellor, N. (2019). Domain wall encoding of discrete variables for quantum annealing and QAOA. *Quantum Science and Technology*, 4(4):045004. ISSN 2058-9565.

Dunning, I., Huchette, J., e Lubin, M. (2017). JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59(2):295–320.

Iosue, J. T. (2022). qubovert. `https://github.com/jtiosue/qubovert`.

Matsuda, Y. (2022). Amplify - fixstars. URL `https://amplify.fixstars.com/en/docs/index.html`.

Tamura, K., Shirai, T., Katsura, H., Tanaka, S., e Togawa, N. (2021). Performance Comparison of Typical Binary-Integer Encodings in an Ising Machine. *IEEE Access*, 9:81032–81039. ISSN 2169-3536.

Zaman, M., Tanahashi, K., e Tanaka, S. (2021). PyQUBO: Python Library for Mapping Combinatorial Optimization Problems to QUBO Form. *arXiv*.