

Trabalho Final

Redes de Computadores - QXD0021



**UNIVERSIDADE
FEDERAL DO CEARÁ**
CAMPUS QUIXADÁ

Heric da Silva Cruz
hericsilvaho@gmail.com

Universidade Federal do Ceará

10 de julho de 2025



Motivação



Algoritmos Genéticos

- Desenvolvidos por John Holland em 1975.
- Inspirados nos processos de evolução natural Trabalham com uma população de soluções candidatas (cromossomos) e usam operadores de seleção, cruzamento e mutação.
- Usados para encontrar soluções aproximadas para problemas NP-Completo e outros.

Algorithm 1 Basic GENETIC ALGORITHM

```
1: initialize population
2: repeat
3:   repeat
4:     crossover
5:     mutation
6:     phenotype mapping
7:     fitness computation
8:   until population complete
9:   selection of parental population
10: until termination condition
```

Pseudocódigo de um algoritmo genético simplificado.

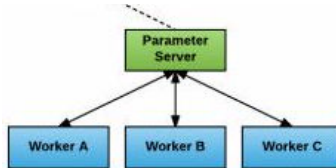
O Desafio da Execução

- A natureza estocástica dos Algoritmos Genéticos significa que cada execução pode gerar uma solução final diferente.
- Para obter um conjunto robusto e **diversificado** de boas soluções para um mesmo problema, é essencial executar o algoritmo **múltiplas vezes**.
- Embora uma única execução possa ter um tempo “razoável”, a necessidade de dezenas ou centenas de execuções sequenciais transforma o processo em uma tarefa extremamente demorada.

A Solução: Kambo-Hive

A solução é dividir as múltiplas execuções do algoritmo, distribuindo a carga de trabalho em uma arquitetura **Cliente-Servidor**:

- **Host (Servidor)**: Gerencia e distribui a fila de tarefas a serem processadas.
- **Workers (Clientes)**: Solicitam tarefas ao Host, executam o algoritmo de forma independente e devolvem o resultado.



O Host distribui tarefas para múltiplos Workers, que as processam em paralelo.

Requisitos do Projeto



Requisitos do Projeto

A Solução foi desenvolvida segundo os requisitos do problema e incluindo os requisitos para a pontuação extra:

- **Disponibilidade do Código:**
Versionamento e publicação no GitHub.
- **Pesquisa Automática de Host:**
Workers encontram o servidor na rede local via broadcast.
- **Distribuição Configurável:**
O Host permite alterar a estratégia de envio de tarefas (FIFO, LIFO, etc).
- **Relatório de Uso Detalhado:**
Geração de um arquivo JSON com estatísticas da execução.

Fluxo de Execução



Passo 1: A Requisição do Worker

O ciclo se inicia com um Worker ativo solicitando uma tarefa ao Host.

- O Worker envia uma mensagem `RequestTask`, que contém apenas seu próprio ID único para identificação.

```
1  #[derive(Debug, Serialize, Deserialize)]
2  pub enum Request {
3      RequestTask { worker_id: Uuid },
4      ReportResult { worker_id: Uuid, result: TaskResult },
5      Heartbeat { worker_id: Uuid },
6  }
```

A enum `Request` define as possíveis ações de um Worker.

Passo 2: A Atribuição da Tarefa

Ao receber a requisição, o Host consulta sua fila de tarefas pendentes e responde conforme a estratégia de distribuição configurada.

- Se houver trabalho, o Host envia a mensagem **AssignTask**.
- Essa mensagem carrega um objeto **Task**, que contém todas as informações que o Worker precisa para executar o algoritmo.

```
1  #[derive(Debug, Clone, Serialize, Deserialize)]
2  pub struct Task {
3      pub id: Uuid,
4      pub graph_id: String,
5      pub run_number: u32,
6      pub ag_config: String,
7  }
8
```

A struct **Task** encapsula os dados de uma única execução.

Passo 3: O Relato do Resultado

Após finalizar o processamento, o Worker envia seu resultado de volta para o Host.

- A mensagem **ReportResult** carrega um objeto **TaskResult**.
- Esta estrutura contém o fitness da solução, o tempo de processamento, e outros detalhes da execução.
- O Host armazena esses dados para o relatório final e confirma o recebimento.

```
1  #[derive(Debug, Clone, Serialize, Deserialize)]
2  pub struct TaskResult {
3      pub task_id: Uuid,
4      pub graph_id: String,
5      pub worker_id: Uuid,
6      pub fitness: f64,
7      pub solution_data: Vec<u8>,
8      pub iterations_run: u32,
9      pub processing_time_ms: u64,
10 }
```

A struct **TaskResult** com os dados da solução encontrada.

O Cérebro do Sistema: O Host

Além de distribuir tarefas, o Host gerencia todo o ciclo de vida da execução através de três componentes principais:

- **TaskManager:**

Controla a fila de tarefas, o status de cada uma (pendente, atribuída, concluída) e implementa as diferentes **estratégias de distribuição** como *FIFO*, *LIFO* e *RANDOM*.

- **ResultAggregator:**

Recebe e armazena todos os `TaskResult` enviados pelos workers, centralizando os dados para a análise final, utiliza um `ArcMutex` para fazer isso.

- **PeriodicSaver:**

De forma assíncrona, salva o progresso dos resultados em um arquivo. Isso garante a **"tolerância a falhas"**, permitindo a recuperação dos dados caso o Host seja interrompido.

Execução



Kambo-Hive em Execução

Host Inicializado

```
Alacrity
hacheric@fedora ~/Projects/kambo-hive «main»
$ cargo build --release
  Finished 'release' profile [optimized] target(s) in 0.00s
hacheric@fedora ~/Projects/kambo-hive «main»
$ ./target/release/kambo-hive-host 192.168.0.11:3005 data/edges report.json random periodic_saver.json 10
```

Host carregando as tarefas e aguardando conexões.

Workers em Execução

```
hacheric@fedora ~/Projects/kambo-hive «main»
$ cargo build --release
  Finished 'release' profile [optimized] target(s) in 0.00s
hacheric@fedora ~/Projects/kambo-hive «main»
$ ./target/release/kambo-hive-worker --auto data/edges
2025-07-09T18:53:35Z INFO kambo_hive::utils Logger inicializado
2025-07-09T18:53:35Z INFO kambo_hive::worker Iniciando descoberta automática de host...
2025-07-09T18:53:35Z INFO kambo_hive::utils Enviando broadcast para encontrar host na porta 2901...
hacheric@fedora ~/Projects/kambo-hive «main»
$ ./target/release/kambo-hive-worker 192.168.0.11:3005 data/edges
```

Workers se conectam, solicitam e processam tarefas em paralelo.

Análise de Desempenho: O Relatório Final

Ao final da execução (ou ao ser interrompido), o Host gera um relatório detalhado em formato JSON, consolidando todos os dados da operação.

Este relatório permite analisar:

- O desempenho individual de cada Worker (tarefas resolvidas, tempo médio)..
- Um resumo geral da saúde da execução (total de tarefas, falhas, etc.).

Conclusão



Conclusão

O projeto **Kambo-Hive** cumpriu todos os requisitos propostos para o trabalho, incluindo os de pontuação extra.

O código-fonte completo, exemplos de uso da biblioteca estão disponíveis no GitHub:

`github.com/hscHeric/kambo-hive`

É importante ressaltar que já existem soluções consolidadas e de alta desempenho para computação científica distribuída (como OpenMP e MPI) o objetivo do projeto é o aprendizado.

Obrigado

