4
Algoritmo Genético com Chaves Aleatórias

Neste capítulo é apresentado um algoritmo genético com chaves aleatórias [9] para o problema min-RWA. A opção por esta classe de algoritmos genéticos é motivada por seu emprego bem sucedido a diversos problemas de otimização combinatória [14, 15, 27, 43, 44, 45]. Além disso, as heurísticas conhecidas para min-RWA baseadas em busca local [48, 49] mostraram-se ineficientes para instâncias de grande porte.

4.1
Visão Geral

Os algoritmos genéticos com chaves aleatórias utilizam uma população de cromossomos que consistem em vetores de números reais com valores no intervalo $[0, 1]$. Cada elemento do vetor é chamado de *chave* e seu valor é gerado aleatoriamente na população inicial. A adaptabilidade (*fitness*, em inglês) do cromossomo é definida pelo custo da solução fornecida por uma heurística que recebe como um de seus dados de entrada o vetor de chaves do cromossomo e devolve uma solução viável para o problema.

O algoritmo de recombinação (ou cruzamento) utilizado foi proposto em [94]. Dados dois cromossomos c_1 e c_2, um novo cromossomo c_{new} é gerado da seguinte forma. Assumindo-se que o cromossomo c_1 é mais apto que o cromossomo c_2, o valor de cada uma das chaves de c_{new} é herdado aleatoriamente da chave correspondente em c_1 com probabilidade p_{rec} e da chave correspondente em c_2 com probabilidade $1 - p_{rec}$, onde p_{rec} é um parâmetro tipicamente fixado em $0,7$. Este algoritmo não faz uso de operadores tradicionais de mutação. Ao invés disso, a cada geração, os piores cromossomos na população são eliminados e substituídos por novos cromossomos gerados aleatoriamente.

A cada nova geração, a população corrente é particionada em três conjuntos: TOP, MID e BOT. Os cromossomos mais aptos são mantidos no conjunto TOP, os menos aptos no conjunto BOT e os restantes no conjunto MID. O tamanho da população e a cardinalidade dos conjuntos TOP, MID e BOT são parâmetros que devem ser ajustados. Como ilustrado na

Figura 4.1, os cromossomos no conjunto TOP são copiados para a próxima geração, e aqueles em BOT são substituídos por novos cromossomos gerados aleatoriamente. Os cromossomos no conjunto MID são substituídos por novos cromossomos obtidos pelo cruzamento de dois cromossomos, sendo que o primeiro é escolhido aleatoriamente do conjunto TOP e o segundo do conjunto $REST = MID \cup BOT$.

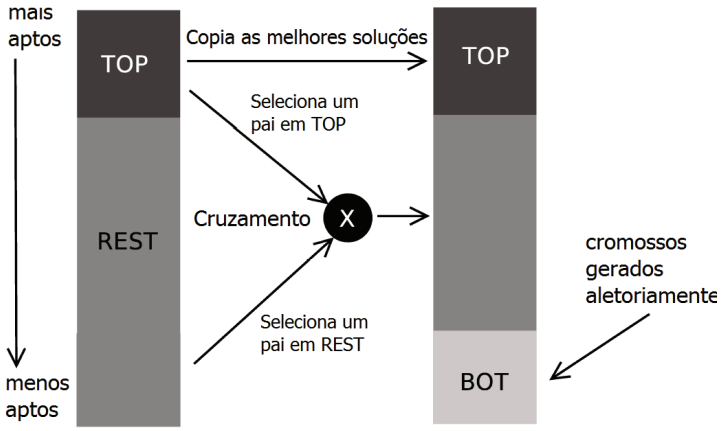

Figura 4.1: Ilustração do processo de transição entre gerações sucessivas.

4.2 Algoritmo Genético para min-RWA

No algoritmo genético com chaves aleatórias para min-RWA, `GA-RWA`, os cromossomos contêm $|\mathcal{T}|$ chaves, cada uma associada a um caminho ótico em $\mathcal{T}$. Define-se $key(i)$ como o valor da chave do caminho ótico $t_i \in \mathcal{T}$ no cromossomo e $sp(i)$ como o tamanho do caminho mais curto entre os nós terminais de t_i em N. A decodificação do cromossomo é realizada em duas etapas. Primeiramente, constrói-se uma ordenação $\pi = [\ \pi(1), \ldots, \pi(|\mathcal{T}|)\ ]$, com $\pi(i) \in \{1, \ldots, |\mathcal{T}|\}$ e $\pi(i) \neq \pi(j)$, para todo $i, j = 1, \ldots, |\mathcal{T}|$, ordenando-se os caminhos óticos $\pi(i)$ em ordem decrescente de $sp(i)$, quebrando-se os empates pelo maior valor de $key(i)$. Ou seja, $\pi(k) = \text{argmax}\{sp(i)*10+key(i) : i \in \{1, \ldots, \mathcal{T}\} \setminus \{\pi(1), \pi(2), \ldots, \pi(k-1)\}\}$. Em seguida, este vetor é passado como parâmetro para a heurística `BF-RWA` (ver Figura 2.6). O número de comprimentos de onda na solução retornada por `BF-RWA` é usado como o valor da adaptabilidade do cromossomo. `GA-RWA` encerra sua execução quando um limite máximo de tempo de execução é atingido ou quando uma solução com custo tão bom quanto um dado valor alvo é encontrada. O tamanho da população e o tamanho dos conjuntos TOP, MID e BOT são ajustados na seção 4.4.

4.3 Framework para Algoritmos Genéticos com Chaves Aleatórias

Algoritmos genéticos com chaves aleatórias foram utilizados para construir heurísticas para diversos problemas de otimização combinatória [14, 15, 27, 43, 44, 45]. Estes algoritmos utilizam os mesmos operadores e a mesma dinâmica populacional apresentados na seção anterior, independentemente do problema a ser resolvido. As principais diferenças entre eles são: a heurística utilizada para fazer a decodificação do cromossomo, o tamanho da população de cromossomos e a cardinalidade dos conjuntos TOP, MID e BOT. A semelhança entre estes algoritmos motivou o desenvolvimento de um *framework* [31, 65] para auxiliar no desenvolvimento de aplicações desta natureza.

Um *framework* é um conjunto de classes, programadas em uma linguagem orientada a objetos, usado para auxiliar no desenvolvimento de software. O *framework* atua onde há funcionalidades em comum a várias aplicações. Todas as funcionalidades em comum são implementadas no *framework*, deixando apenas os detalhes específicos de cada aplicação a serem implementados pelo desenvolvedor. As classes que implementam as funcionalidades em comum a todas as aplicações são chamadas de pontos fixos e aquelas que implementam os detalhes específicos são chamadas de pontos flexíveis.

O objetivo do *framework* proposto neste trabalho é auxiliar no desenvolvimento de algoritmos genéticos com chaves aleatórias para problemas de otimização combinatória. Ele é baseado em diversos trabalhos sobre reutilização de software para resolver problemas de otimização combinatória [4, 5, 6, 19, 36, 73, 90]. O conceito em comum nesses trabalhos é reutilizar somente o código que gerencia a dinâmica dos algoritmos, deixando todo o código que é dependente do problema que está sendo resolvido para ser implementado pelo desenvolvedor.

O diagrama de classes do *framework* é apresentado na Figura 4.2, usando a linguagem UML [61]. Neste diagrama, visualizam-se o ponto flexível do framework (em destaque), responsável pela decodificação e avaliação do cromossomo, e os pontos fixos, responsáveis por toda a dinâmica do algoritmo genético com chaves aleatórias.

A classe `BibRand` implementa o gerador de números aleatórios que é utilizado pelos operadores de mutação e recombinação. As classes abstratas `Mutation`, `Crossover` e `Evaluator` definem a interface dos métodos que implementam os operadores de mutação, recombinação e avaliação da aptidão dos cromossomos, respectivamente.

A classe `Random` implementa o método virtual `Execute(vector<double>& c)` declarado na classe abstrata `Mutation`. Este método recebe como

parâmetros o vetor de chaves de um cromossomo e aplica um operador de mutação aleatório neste cromossomo. O construtor da classe `Random` recebe como parâmetro o número de chaves do cromossomo.

A classe `Biased` implementa o método virtual `Execute(vector<double> c1, vector<double> c2, vector<double>& new)` declarado na classe abstrata `Crossover`. Este método recebe como parâmetros os vetores de chaves de dois cromossomos e retorna um novo cromossomo, gerado a partir da aplicação do operador de recombinação baesiano [94] descrito no início deste capítulo. O construtor da classe `Biased` recebe como parâmetros o tamanho do cromossomo e a probabilidade p_{rec} do valor de cada chave do novo cromossomo ser herdado da chave correspondente em `c1`. Analogamente, as chaves do novo cromossomo são herdadas de `c2` com probabilidade $1 - p_{rec}$.

A classe `Decoder` implementa o método virtual `Fitness(vector <double>& c)` definido na classe abstrata `Evaluator`. Este método recebe como parâmetro o vetor de chaves de um cromossomo e calcula o valor da adaptabilidade do cromossomo. O construtor da classe `Decoder` recebe como parâmetro o nome do arquivo que contêm os dados da instância do problema. `Decoder` é um ponto flexível do *framework*. Ele pode ser substituída por qualquer heurística para qualquer problema de otimização que implemente a função `Fitness(vector<double>& c)` definida em `Evaluator`.

A classe `GeneticAlgorithm` gerencia a interação entre as classes mencionadas anteriormente. Sua função é aplicar os operadores de mutação, recombinação, avaliação e seleção da forma como descrita no início deste capítulo. O construtor desta classe recebe como parâmetros o número de chaves aleatórias no cromossomo e o número de cromossomos nos conjuntos TOP, MID e BOT, além de três ponteiros para objetos do tipo `Mutation`, `Crossover` e `Evaluator`, respectivamente. Esta classe implementa a função `Execute(int max_generations, int target_fitness, int time_limit)` que executa o algoritmo genético até que `max_generations` gerações tenham sido avaliadas, ou o limite máximo de tempo `time_limit` tenha sido atingido, ou um cromossomo cujo valor da função de avaliação seja menor ou igual ao alvo `target_fitness` tenha sido encontrado.

O *framework* foi implementado na linguagem C++ versão 4.0.3. A implementação em C++ da heurística `GA-RWA`, descrita na seção anterior, utilizando o *framework* é apresentada na Figura 4.3. O procedimento recebe como parâmetros o nome do arquivo de texto (`filename`) contendo os dados da instância do problema, o número máximo de gerações (`max_generations`), o valor do alvo (`target_cost`), o tempo máximo de execução (`time_limit`), o número de chaves aleatórias no cromossomo (`chromosome_size`) e o número

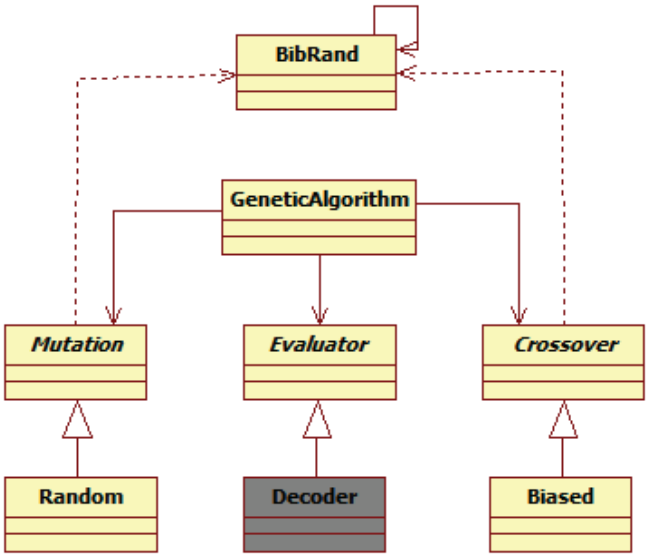

Figura 4.2: Diagrama de classes do *framework* para algoritmos genéticos com chaves aleatórias.

de cromossomos nos conjuntos TOP (`size_top`), MID (`size_mid`) e BOT (`size_bot`), além da semente do gerador de números aleatórios (`random_seed`). Na linha 4, o gerador de números aleatórios é inicializado. Os operadores de mutação e recombinação são inicializados nas linha 5 e 6, respectivamente. A classe `BfRWA`, herdada da classe `Evaluator`, é instanciada como operador de avaliação da adaptabilidade dos cromossomos na linha 7. O algoritmo genético é inicializado na linha 8 e executado na linha 10. A execução é interrompida quando o número máximo de gerações (`max_generations`) for atingido, ou quando uma solução com custo tão bom quanto o alvo (`target_cost`) for encontrada, ou quando o limite máximo de tempo de execução (`time_limit`) for atingido. Por fim, o melhor cromossomo encontrado durante toda a execução do algoritmo é decodificado e a solução correspondente é exibida ao usuário.

4.4 Experimentos Computacionais

As heurísticas 2-`EDR+TS-PCP` [70], `MS-BFD` (descrita no Capítulo 3) e `GA-RWA` foram implementadas em C++ e compiladas com o compilador GNU GCC versão 4.0.3. O primeiro experimento descrito abaixo foi realizado em um Intel Itanium 2 com 1.5 GHz, enquanto o segundo foi realizado em um Pentium IV 3.4 GHz.

O conjunto de instâncias utilizado é composto pelas instâncias mais difíceis dentre as 112 instâncias nos conjuntos Y, Z e W, selecionadas

```
1.  void GA-RWA(string filename, int max_generations, int target_cost,
2.              int time_limit, int chromosome_size, int size_top,
3.              int size_mid, int size_bot, int random_seed) {
4.    BibRand::Initialize(random_seed);
5.    Mutation* mutation = (Mutation*) new Random(chromosome_size);
6.    Crossover* crossover = (Crossover*) new Biased(chromosome_size, 0.7);
7.    Evaluator* bf_rwa = (Evaluator*) new BfRWA(filename);
8.    GeneticAlgorithm ga(chromosome_size, size_top, size_mid,
9.                        size_bot, mutation, crossover, bfd_rwa);
10.   ga.Execute(max_generations, target_cost, time_limit);
11.   bfd_rwa->Output(ga.cromossome(0));
12. }
```

Figura 4.3: Exemplo de aplicação do *framework* de algoritmos genéticos para min-RWA.

da seguinte forma. Primeiramente, cinco execuções de `MS-BFD` com 10.000 iterações foram realizadas variando-se as sementes do gerador de números aleatórios. O custo da melhor solução encontrada nas cinco execuções foi definido como o valor alvo para a instância. Em seguida, selecionaram-se as instâncias para as quais `MS-BFD` não encontrou uma solução com custo igual ao valor alvo nas primeiras 1.000 iterações em alguma das cinco execuções. Trinta instâncias foram selecionadas: 14 do conjunto Y, 10 do conjunto Z e 6 do conjunto W. Vale a pena salientar que as instâncias restantes não são necessariamente fáceis, pois o valor alvo pode ainda estar longe do custo da solução ótima.

O primeiro experimento avalia e compara o desempenho das heurísticas `MS-BFD` e `GA-RWA`. Seis versões da heurística `GA-RWA` foram experimentadas, cada uma delas com diferentes valores para o tamanho da população e o número de cromossomos nos conjuntos TOP, MID e BOT. As três primeiras versões (`V1`, `V2` e `V3`) trabalham com uma população de $n = |X|$ cromossomos. As outras três versões (`V4`, `V5` e `V6`) trabalham com uma população de $2 \times n$ cromossomos. As versões `V1` e `V4` têm $|TOP| < |BOT|$ (com $|TOP| = 0.01 \times n$ e $|BOT| = 0.02 \times n$), enquanto `V2` e `V5` têm $|TOP| > |BOT|$ (com $|TOP| = 0.25 \times n$ e $|BOT| = 0.05 \times n$). As outras versões `V3` e `V6` têm $|TOP| = |BOT| = 0.15 \times n$.

A heurística `MS-BFD` e as seis versões de `GA-RWA` foram executadas 200 vezes em cada uma das instâncias selecionadas, variando-se a semente do gerador de número aleatórios. As heurísticas foram configuradas para parar quando uma solução com custo menor ou igual ao valor alvo fosse encontrada. Os resultados computacionais para este experimento são relatados na Tabela 4.1. A primeira coluna apresenta o nome da instância. A coluna seguinte mostra o desvio relativo ([alvo − LB]/LB) entre o valor alvo e o

limite inferior LB para o custo da solução ótima de cada instância, dado pelo valor ótimo da relaxação linear da formulação (2-1) a (2-4), apresentada no Capítulo 2. A terceira coluna exibe o tempo médio (em segundos) gasto por `MS-BFD` para encontrar uma solução tão boa quanto o alvo. O mesmo resultado para as seis versões de `GA-RWA` são apresentados nas seis últimas colunas como um desvio percentual dos tempos correspondentes de `MS-BFD`. Dentre as seis versões de `GA-RWA`, aquela que apresentou melhor desempenho foi `V2`. Na média, esta variante encontra soluções tão boas quanto o alvo em 77% do tempo de `MS-BFD`. Para a instância com alvo mais difícil (Z.4×25.60), `MS-BFD` demorou em média 13.583 segundos para atingir o alvo, enquanto `GA-RWA` demorou apenas 8.112 segundos. Dentre as 30 instâncias testadas, `MS-BFD` foi mais rápido que `V2` em apenas duas instâncias (Z.10×10.20 e Finland). Considerando-se apenas as instâncias do conjunto W de instâncias realistas, os tempos para `V2` atingir o alvo foram em média dois terços dos tempos de `MS-BFD`. Sendo assim, está versão foi adotada como a versão padrão de `GA-RWA`.

Os mesmos resultados podem ser observados nas Figuras 4.4 a 4.9, onde são traçados os gráficos da distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo em função do tempo para seis das instâncias apresentadas na Tabela 4.1. Os gráficos foram traçados utilizando-se a metodologia descrita em [1, 2]. Primeiramente, os tempos de execução de cada heurística são ordenados. Em seguida, associa-se ao i-ésimo menor tempo τ_i uma probabilidade $\phi_i = (i - \frac{1}{2})/200$. Por fim, traça-se os pontos $b_i = (\tau_i, \phi_i)$, para $i = 1, \ldots, 200$. Pode-se observar que a probabilidade de `GA-RWA` atingir o alvo é maior que a de `MS-BFD` em todas as instâncias apresentadas nas Figuras 4.4 a 4.9, exceto Z.10×10.20 (Figura 4.7). Entretanto, observa-se que neste último os resultados das duas heurísticas estão muito próximos, enquanto nas outras cinco instâncias os resultados `GA-RWA` são bem superiores aos de `MS-BFD`.

O segundo experimento avalia e compara o desempenho das heurísticas `2-EDR+TS-PCP` e `GA-RWA`. Para cada instância, a Tabela 4.2 exibe o nome, o limite inferior para o custo da solução ótima e o desvio relativo mínimo, médio e máximo das soluções obtidas com `2-EDR+TS-PCP` e `GA-RWA` em dez minutos de execução. Cada heurística foi executada cinco vezes para cada instância com diferentes sementes para o gerador de números aleatórios. Os desvios relativos das soluções retornadas por `GA-RWA` são menores que os obtidos por `2-EDR+TS-PCP` em 29 das 30 instâncias testadas. O desvio relativo médio sobre todas as instâncias na Tabela 4.2 é 9,3% para `GA-RWA` e 17,1% para `2-EDR+TS-PCP`. `GA-RWA` encontra melhores soluções que `2-EDR+TS-PCP` principalmente nas maiores instâncias. Isto se deve ao fato de que quanto maior

Tabela 4.1: Tempos médios obtidos por MS-BFD e diferentes versões de GA-RWA para encontrar uma solução com custo tão bom quanto o alvo.

			GA-RWA/MS-BFD (%)					
Instância	alvo (%)	MS-BFD (s)	V1	V2	V3	V4	V5	V6
Y.4.20.4	5,3	118,0	97,2	96,4	99,3	90,5	91,0	90,4
Y.3.40.5	11,3	2913,4	86,8	75,3	81,7	84,0	90,8	81,7
Y.3.60.5	11,7	1307,4	92,8	91,8	95,0	98,3	88,6	95,8
Y.4.60.5	18,4	233,6	96,7	97,2	103,0	86,9	103,6	100,6
Y.5.60.1	9,1	2262,7	90,3	73,6	80,3	86,9	68,8	74,2
Y.3.80.1	15,1	3036,2	87,9	78,3	102,8	97,3	90,2	93,0
Y.3.80.5	8,7	632,4	89,4	84,3	87,3	95,3	87,9	99,9
Y.4.80.1	55,3	194,1	75,8	78,3	78,9	89,0	88,1	85,8
Y.4.80.5	15,4	981,1	73,4	62,4	84,7	82,9	88,0	83,5
Y.5.80.1	9,3	3066,4	62,8	41,8	59,0	52,2	42,1	50,5
Y.5.80.2	0,0	8538,7	66,8	53,1	60,8	63,1	45,4	59,8
Y.4.100.1	18,4	238,3	93,8	89,1	98,9	75,8	80,7	84,3
Y.5.100.1	5,5	754,7	81,5	80,1	77,4	86,8	79,7	81,2
Y.5.100.2	1,4	1330,4	62,2	54,3	64,7	64,7	62,9	67,2
Z.10×10.20	14,8	329,6	120,5	118,5	119,0	110,6	102,2	98,2
Z.6×17.40	3,6	388,8	104,3	82,9	98,2	89,0	83,8	95,1
Z.4×25.60	1,6	13583,2	65,8	59,8	70,3	67,3	68,1	65,8
Z.10×10.60	13,0	1282,3	90,1	94,1	95,7	91,4	90,6	92,9
Z.4×25.80	1,2	739,9	76,4	72,7	78,2	75,3	86,5	82,7
Z.5×20.80	2,0	145,6	81,3	77,1	86,6	89,6	91,7	87,2
Z.6×17.80	2,9	422,8	69,5	72,5	70,9	76,8	73,7	75,2
Z.8×13.80	3,9	604,9	94,1	76,4	84,6	91,5	78,6	98,6
Z.10×10.80	11,7	6522,1	89,7	94,7	94,1	90,0	94,2	83,6
Z.5×20.100	2,4	3181,5	101,4	96,6	89,4	102,3	105,9	112,5
Finland	0,0	237,0	89,3	101,7	100,0	94,7	98,6	95,2
ATT	20,0	24,9	74,9	64,8	63,6	65,7	63,4	65,9
ATT2	0,0	1584,6	62,2	54,5	63,7	72,4	53,1	73,0
NSF.3	0,0	355,7	110,5	78,8	86,3	81,9	78,0	87,7
NSF.12	2,6	32,8	97,2	62,6	70,4	57,4	50,3	58,1
NSF2.12	0,0	1370,3	83,5	46,4	64,0	52,3	31,9	42,9
Média:	8,8	1880,4	85,6	77,0	83,6	82,1	78,6	82,1

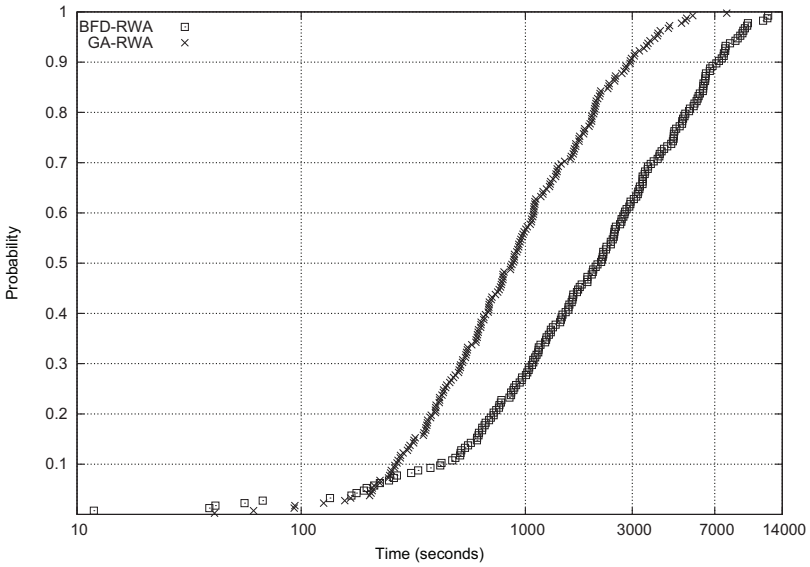

Figura 4.4: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância Y.5.80.1.

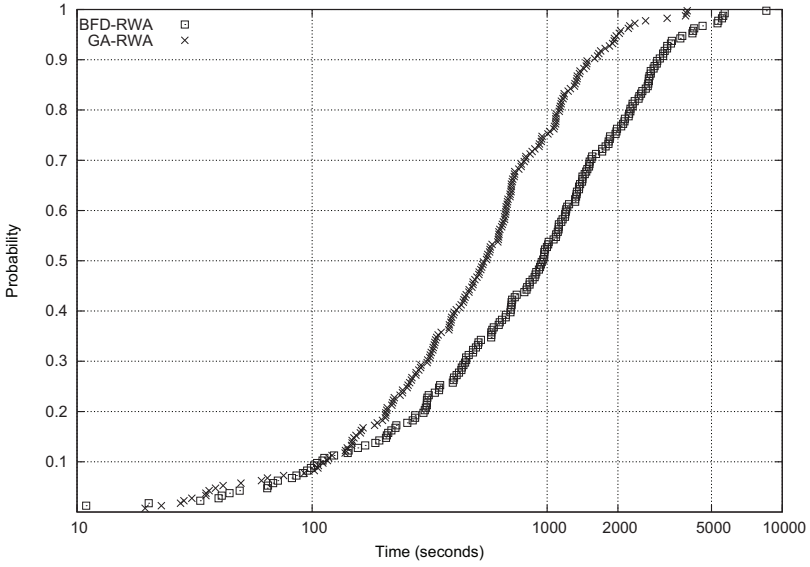

Figura 4.5: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância Y.5.100.2.

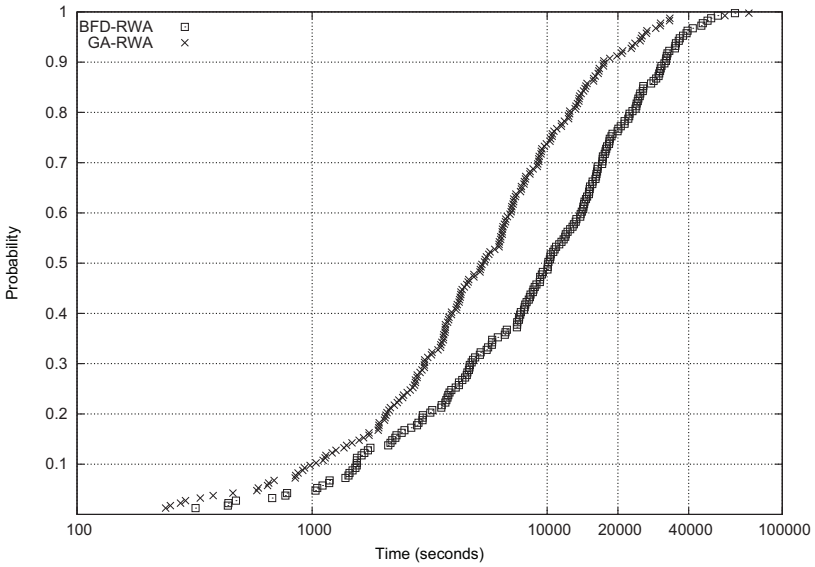

Figura 4.6: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância Z.4x25.60.

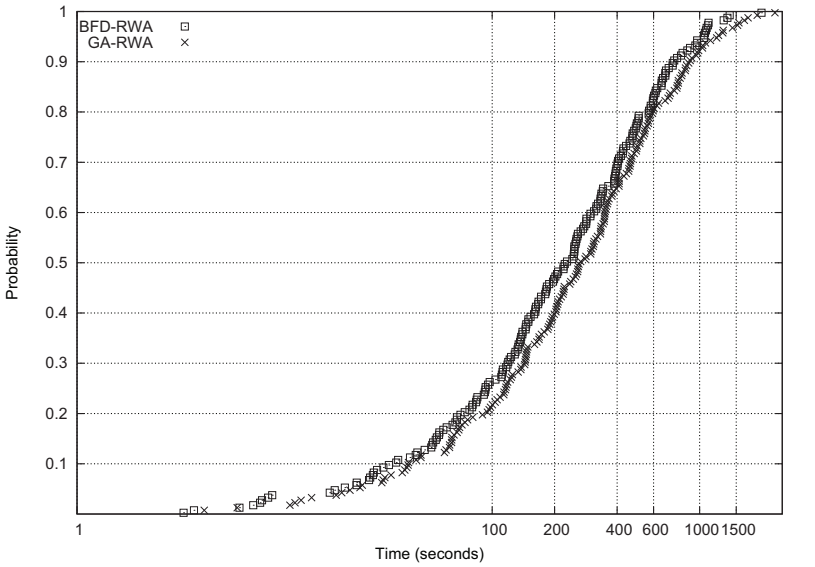

Figura 4.7: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância Z.10x10.20.

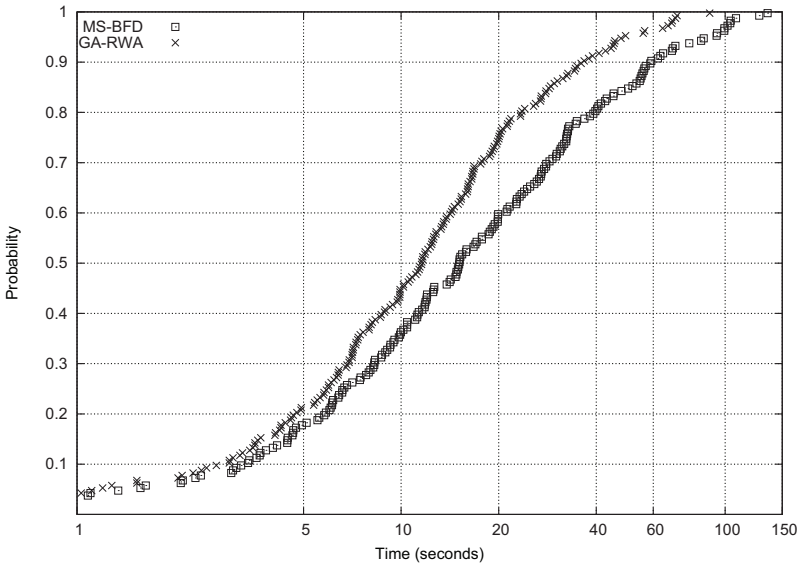

Figura 4.8: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância ATT.

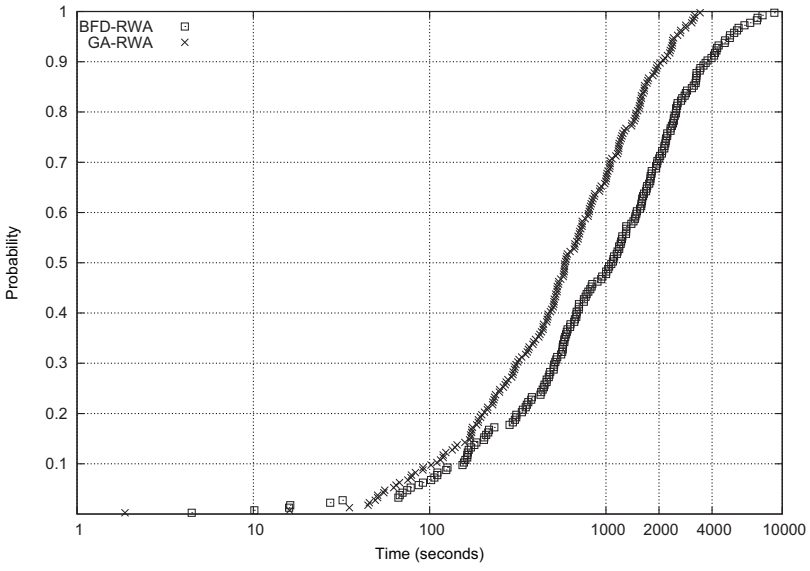

Figura 4.9: Distribuição empírica de probabilidade das heurísticas `MS-BFD` e `GA-RWA` atingirem o alvo para a instância ATT2.

o número de caminhos óticos na instância, maior é o tamanho do problema de coloração de partições que deve ser resolvido na segunda fase de `2-EDR+TS-PCP`. As instâncias nos conjuntos Y e Z têm até 9900 requisições de caminhos óticos. Como o procedimento 2-EDR seleciona duas rotas alternativas para cada caminho ótico, o grafo de conflitos que deve ser colorido pela heurística TS-PCP tem até 19.800 nós. Considerando-se apenas as instâncias do conjunto W, que são bem menores que aquelas dos conjunto Y e Z, o desvio relativo médio das instâncias retornadas por `2-EDR+TS-PCP` é 4.5%, enquanto o mesmo valor para `GA-RWA` é 4.0%.

Neste capítulo apresentou-se um algoritmo genético com chaves aleatórias para min-RWA. Experimentos computacionais mostraram que este algoritmo foi capaz de melhorar os resultados do algoritmo de multi-partida `MS-BFD` e da heurística `2-EDR+TS-PCP` proposta em [70]. Também pode-se observar a robustez do algoritmo genético dado que todas as versões de `GA-RWA` (usando diferentes configuração de parâmetros) obtiveram resultados semelhantes. O algoritmo genético em conjunto com o limite inferior calculado por programação linear permitiu provar a otimalidade de 41% das instâncias de teste.

Tabela 4.2: Desvio relativo das soluções encontradas por `GA-RWA` e `2-EDR+TS-PCP` em 10 minutos de execução.

		`2-EDR+TS-PCP` (%)			`GA-RWA` (%)		
Instância	LB	mínimo	média	máximo	mínimo	média	máximo
Y.4.20.4	19	10,5	13,7	15,8	5,3	6,3	10,5
Y.3.40.5	53	13,2	14,7	15,1	11,3	12,8	13,2
Y.3.60.5	77	13,0	15,1	16,9	11,7	12,5	13,0
Y.4.60.5	49	24,5	25,3	26,5	18,4	18,4	18,4
Y.5.60.1	33	18,2	21,2	24,2	9,1	9,7	12,1
Y.3.80.1	106	17,9	18,7	20,8	15,1	15,5	16,0
Y.3.80.5	104	12,5	12,7	13,5	8,7	8,8	9,6
Y.4.80.1	47	63,8	65,1	68,1	55,3	55,3	55,3
Y.4.80.5	65	21,5	22,2	23,1	15,4	16,0	16,9
Y.5.80.1	43	20,9	21,9	23,3	9,3	11,2	11,6
Y.5.80.2	59	6,8	8,1	8,5	1,7	1,7	1,7
Y.4.100.1	76	38,2	40,0	43,4	18,4	18,4	18,4
Y.5.100.1	55	23,6	30,5	36,4	5,5	5,5	5,5
Y.5.100.2	73	16,4	21,1	30,1	1,4	1,6	2,7
Z.10×10.20	27	22,2	25,2	25,9	14,8	15,6	18,5
Z.6×17.40	84	7,1	7,6	8,3	3,6	4,0	4,8
Z.4×25.60	192	5,7	5,9	6,3	1,6	2,0	2,1
Z.10×10.60	77	20,8	21,3	22,1	13,0	13,2	14,3
Z.4×25.80	257	9,7	10,5	11,3	1,2	1,3	1,6
Z.5×20.80	205	14,6	15,5	17,6	2,0	2,0	2,0
Z.6×17.80	171	15,2	16,8	19,3	2,9	3,0	3,5
Z.8×13.80	129	14,7	15,8	17,1	3,9	3,9	3,9
Z.10×10.80	103	26,2	27,6	29,1	11,7	12,4	12,6
Z.5×20.100	250	2,8	8,6	15,6	2,8	2,8	2,8
Finland	46	2,2	2,2	2,2	0,0	0,4	2,2
ATT	20	10,0	11,0	15,0	20,0	20,0	20,0
ATT2	113	0,9	1,1	1,8	0,0	0,0	0,0
NSF.3	22	4,5	4,5	4,5	0,0	0,9	4,5
NSF.12	38	2,6	4,2	7,9	2,6	2,6	2,6
NSF2.12	35	2,9	4,0	5,7	0,0	0,6	2,9
	Média:	15,4	17,1	19,2	8,9	9,3	10,1