N-rainhas com Busca Tabu - Otimização Heurística

Cristyan Lisbôa

***Resumo*—Este trabalho aborda o problema de N-rainhas através da teoria de otimização heurística, em particular, a partir do algoritmo de Busca Tabu. Utilizando uma formulação combinatorial, o objetivo é desenvolver uma implementação simples capaz de dispor N rainhas em um tabuleiro de xadrez $N \times N$ sem que ocorram ataques entre elas. O método utiliza uma codificação inteira na qual as posições das rainhas são convertidas em elementos de um vetor através de um mapeamento simples. Por último, são realizadas avaliações numéricas através do *software* MATLAB visando comparar o desempenho do algoritmo com os mecanismos propostos.**

Palavras-chaves: **N-rainhas, Otimização Heurística, Busca Tabu, MATLAB.**

I. Introdução

O problema conhecido como N-rainhas, na realidade, consiste em uma generalização do problema das 8-rainhas originalmente proposto por Max Bezzel em 1848. O objetivo é alocar N rainhas em um tabuleiro de xadrez de dimensão $N \times N$ de modo que não ocorram conflitos (ataques) entre elas. Quando isso ocorre, então uma solução ótima é encontrada e o número de ataques é zero. Assim, elas não podem compartilhar uma mesma linha, coluna ou diagonal. Desde sua concepção, diversas estratégias foram propostas para sua resolução, por exemplo, abordagens envolvendo algoritmos genéticos, redes neurais, *backtracking*, Busca Tabu, entre outros [1].

Os autores Martinjak e Golub [2] realizaram um estudo comparativo entre três algoritmos de otimização heurística para resolver as N-rainhas. Os resultados apontaram que tanto o algoritmo genético quanto o recozimento simulado convergem rapidamente para próximo da solução e gastam a maior parte tempo em pequenos decrementos da função objetivo. Já a Busca Tabu, apesar do custo computacional superior, realiza menos iterações para encontrar uma solução. Em [3], é utilizado o algoritmo de colônia de formigas para resolver as N-rainhas, já em [4] é implementado o algoritmo de *backtracking*. Além disso, a referência [5] apresenta um novo operador de mutação, baseado na seleção aleatória das rainhas envolvidas em conflito, para uma implementação de algoritmo genético aplicado as N-rainhas.

Quanto ao algoritmo de Busca Tabu, ele é pode entendido como um método iterativo de busca local (em vizinhança) capaz de evitar mínimos locais através de operadores de memória e lista de movimentos proibidos. Uma característica interessante, assim como acontece no recozimento simulado, é que a Busca Tabu é classificada como um método de solução única que aceita movimentos na direção de degradação da função objetivo. Isso é justificado pois a solução incumbente pode ser armazenada em separado e a solução corrente deve percorrer o espaço de busca sem que haja retorno para soluções previamente visitadas [6]. Ao longo das últimas décadas, a Busca Tabu demonstrou ser capaz de resolver diversos problemas combinatoriais clássicos, tais como: roteamento, caixeiro viajante, N-rainhas, coloração de grafos, mochila, entre outros.

A partir disso, a proposta deste trabalho é abordar o problema das N-rainhas através do algoritmo de Busca Tabu. Além de implementar as estratégias essenciais que definem o método (lista Tabu e critério de aspiração), o objetivo consiste em aplicar um critério de diversificação quando houver estagnação da solução corrente por um certo número de iterações. Ademais, também deve ser avaliado a influência de duas técnicas de geração de vizinhos, ambas baseadas na permuta entre elementos de um certo vetor, no desempenho do algoritmo. Como dados de entrada, o usuário deve informar o número de rainhas, o tipo de vizinhança e de solução inicial (fixa ou aleatória) tendo como saída uma solução para o problema de N-rainhas. Além disso, faz parte do escopo deste artigo descrever o problema em estudo e sua representação, características do algoritmo utilizado e resultados numéricos de sua implementação no MATLAB.

II. Apresentação do problema

No problema das N-rainhas o objetivo é determinar as posições de N rainhas em um tabuleiro com N linhas e N colunas sem que ocorram ataques entre elas. Isto é satisfeito quando elas não compartilham de uma mesma linha, coluna ou diagonal. A Figura 1, por exemplo, ilustra duas situações diferentes em que os quadrados na cor preta representam as rainhas. Na esquerda, é possível observar que existem ataques entre as rainhas uma vez que estão localizadas em duas diagonais. Este caso pode representar a solução inicial ou a solução corrente de uma determinada iteração do algoritmo. Já na direita, nota-se que não existem cruzamentos entre elas sendo uma solução do problema no caso particular de $N = 7$.

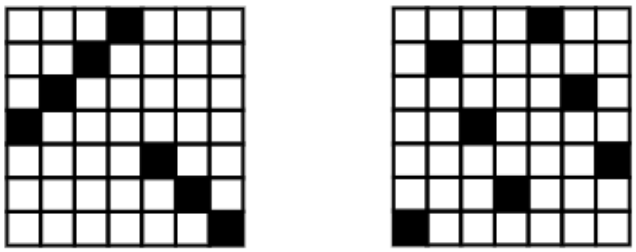

Figura 1. Exemplos de solução do problema.

Neste ponto é oportuno destacar que o tamanho do espaço de busca está diretamente associado a dois aspectos: o primeiro, e mais evidente, é quantidade de rainhas (quanto

Escola de Engenharia da Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Rio Grande do Sul, Brasil (e-mail: {cristyan.lisboa}@ufrgs.br).

mais rainhas maior o espaço de busca). Já o segundo está relacionado ao modelo otimização adotado para resolver o problema. No caso da representação binária, as posições do tabuleiro são variáveis binárias que indicam a presença ou não de rainhas. Assim, um problema com N rainhas terá N^2 variáveis de decisão binária e espaço de busca de tamanho $C_N^{N^2}$ (combinação de N^2 elementos tomados N a N). Por outro lado, na representação combinatorial (adotada neste trabalho) a posição das rainhas segue o padrão demonstrado na Figura 2. O vetor S_k, que representa a solução corrente da iteração k, é composto por $N = 8$ elementos cujo valor indica a coluna e a posição indica linha da rainha (destacado pelos números na cor vermelha). Dessa forma, um problema com N rainhas tem N variáveis de decisão inteira e espaço de busca $N!$ (N fatorial).

$$S_k \begin{bmatrix} 3 & 1 & 4 & 8 & 5 & 7 & 2 & 6 \\ 1 & 2 & 3 & 4 & 5 & 7 & 8 & 9 \end{bmatrix}$$

Figura 2. Modelo combinatorial.

Um aspecto que vale a pena ser destacado é que em função do modelo utilizado as soluções podem ser obtidas pela permuta simples entre os elementos do vetor S_k. Assim, no momento de determinar a vizinhança da solução S_k, isto é, do vetor linha de N posições, basta determinar $N(N-1)/2$ vizinhos. Além disso, esta representação impede a ocorrência de conflitos em uma mesma linha ou coluna, sendo o número de ataques calculado aqueles que acontecem nas diagonais.

No intuito de computar o número de ataques, ou seja, da função objetivo a ser minimizada a ideia é a seguinte: para cada elemento do vetor S_k, calcular diferença entre sua posição e seu valor. Um ataque deve ser registrado quando o resultado desta operação for igual entre dois elementos distintos. O mesmo procedimento deve ser realizado considerando a operação de soma. Observe, na Figura 3, que elementos de uma mesma diagonal, tanto na esquerda quanto na direita, apresentam valor constante e por isso o valor da função objetivo pode ser determinado.

0	−1	−2	−3	−4	−5	−6
1	0	−1	−2	−3	−4	−5
2	1	0	−1	−2	−3	−4
3	2	1	0	−1	−2	−3
4	3	2	1	0	−1	−2
5	4	3	2	1	0	−1
6	5	4	3	2	1	0

2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12
7	8	9	10	11	12	13
8	9	10	11	12	13	14

Figura 3. Cálculo da função objetivo.

III. Algoritmo de Busca Tabu

Em 1986, poucos anos após Kirkpatrick e colegas terem desenvolvido a algoritmo de recozimento simulado, Glover propôs o algoritmo de Busca Tabu. O princípio básico é permitir ao procedimento de busca local escapar de mínimos locais utilizando uma estrutura de memória e lista de movimentos proibidos [7], [8].

A. Solução inicial

A etapa de solução inicial, tal como o nome sugere, é aquela em o ponto de partida deve ser constituído. No contexto deste trabalho foram utilizados dois métodos distintos. O primeiro, é através da geração aleatória, na qual a solução inicial é gerada através do comando *randperm*. O segundo método é a solução inicial fixa, na qual o usuário deve informar um vetor linha de N elementos. Esta segunda opção foi mais utilizada durante a fase de verificação (*debud*) do código.

B. Função objetivo

A função objetivo considerada neste trabalho é o número de conflitos entre as rainhas. Assim, quanto menor for o número de ataques mais próximo de uma resposta encontra-se a solução corrente. O cômputo da função objetivo identifica os ataques através da quantidade de números iguais que resultam das operações de diferença e soma do vetor S_k previamente explicados. Neste ponto, é importante destacar que são investigados dois métodos para calcular o número de conflitos.

O primeiro é o mais simples e considera os ataques apenas entre rainhas adjacentes. Para exemplificar, note que a Figura 1 na esquerda apresenta dois blocos de rainhas na mesma diagonal (um superior e outro inferior). No superior quatro rainhas estão na mesma diagonal constituindo três ataques. Já o inferior apresenta três rainhas constituindo dois ataques. Portanto, o número de conflitos é o somatório dos ataques dos grupos de rainhas mais eventuais cruzamentos. Ademais, o número de ataques de cada grupo é igual ao número de rainhas do grupo menos uma unidade.

O segundo considera os ataques entre rainhas não adjacentes. No caso da Figura 1 na esquerda, o bloco superior contém seis ataques e o inferior contém três ataques. O número total é determinado da mesma maneira que no caso anterior, sendo que a diferença está no cálculo de ataques dos grupos. Em particular, é possível concluir que o número de ataques de um grupo é o resultado da combinação do número de rainhas do grupo tomadas dois a dois, ou seja,

$$f_g = C_2^{n_g} = \frac{n_g!}{(n_g-2)!\,2!} \tag{1}$$

onde f_g é número de ataques do grupo g.

Comentário 1: Apesar de simples entendimento e implementação, não foi possível determinar uma expressão analítica capaz de computar o número de conflitos. Para isso, uma possibilidade a ser investigada seria abordar o problema sob a perspectiva da teoria de grafos, tal como acontece no clássico problema do caixeiro viajante.

C. Vizinhança

A geração de vizinhos ao longo deste trabalho ocorre através da permutação simples entre os elementos da solução corrente S_k. Por exemplo, suponha que $S_k = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$, então os $3(3-1)/2 = 3$ vizinhos são os seguintes:

- $S_k^1 = \begin{bmatrix} 2 & 1 & 3 \end{bmatrix}$
- $S_k^2 = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$

- $S_k^3 = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$

Contudo, uma alternativa que também foi implementada trata-se da geração de vizinhos por permuta até encontrar um vizinho melhor que a solução corrente. A ideia é que o número de operações realizadas seja reduzido e o método avance mais rapidamente em instâncias pequenas. Para ilustrar, suponha que o vizinho S_k^3 seja o melhor entre os três e o vizinho S_k^1 seja melhor apenas que a solução corrente. Neste caso, ao determinar o número de ataques $f(S_k^1)$ o cálculo da vizinhança pode ser encerrado.

D. Lista Tabu

A finalidade da lista Tabu é evitar o comportamento de ciclagem, ou seja, que a solução corrente retorne para uma solução previamente visitada sem impedir a descoberta de soluções de maior qualidade. Este é o elemento de memória de curto prazo [7] sendo implementado na forma de uma matriz quadrada de ordem N. Os elementos da diagonal principal não são utilizados, já os da matriz trigular superior são utilizados para armazenar os movimentos que são tabus, enquanto que os da matriz triangular inferior são destinados ao critério de diversificação. Em cada iteração, sempre que o melhor vizinho é selecionado (e não é tabu), a solução corrente é substituída e um tabu é marcado. Portanto, durante algumas iterações seguintes o movimento oposto não pode ser executado a menos que o critério de aspiração seja satisfeito.

A duração dos tabus foi definida a partir da soma entre a iteração corrente k com um número inteiro aleatório D que está entre um valor mínimo D_{min} e um valor máximo D_{max}. O movimento deixa de ser tabu quando o algoritmo avança de modo que a iteração corrente supera este valor. Por fim, vale mencionar que este é o único mecanismo que insere aleatoriedade no método, pois caso os tabus fossem de duração fixa, então o método seria determinístico.

E. Critério de aspiração

O critério de aspiração estabelece condições sob os quais os tabus podem ser ignorados [8]. O critério mais comum (e que foi implementado neste trabalho) é a aspiração quando a solução visitada tem valor da função objetivo melhor que o da solução incumbente. No caso deste artigo, um movimento da lista Tabu é aceito apenas quando o número de ataques é inferior ao da melhor solução até então.

F. Mecanismo de diversificação

No mecanismo de diversificação, tal como o nome sugere, o propósito é fazer com que o método avance para regiões não exploradas do espaço de solução quando certas condições são verificadas [8]. Por isso ele também é conhecido como memória de longo prazo, já que deve guardar quais foram as soluções previamente visitadas. No caso deste trabalho, a ideia é utilizar a matriz triangular inferior da matriz de tabus para identificar os movimentos ainda não executados. Então, a partir da estagnação da solução corrente durante n_{est} iterações, uma sequência de n movimentos não realizados é aplicada na solução corrente.

A Figura 4 ilustra a ideia central do critério de diversificação implementado. As bolas em azul representam regiões previamente visitadas do espaço de solução, enquanto a bola vermelha representa a região que contém uma solução ótima. Em cinza está a região explorada na iteração corrente e após a aplicação da diversificação. Suponha que o algoritmo tenha explorado toda a vizinhança corrente e não encontrou um vizinho com número de ataques inferior. Então aplica-se a diversificação e a solução passa a ser S_{k+n}. Novamente, a vizinhança é percorrida e nenhum vizinho melhor é encontrado. Finalmente, após aplicar o critério de diversificação, a solução vai para a região em vermelho e encontra a solução ótima em poucas iterações.

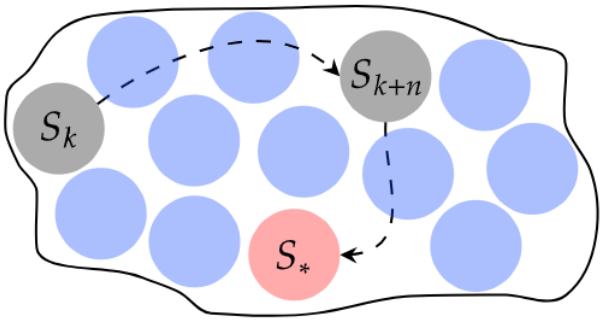

Figura 4. Espaço de solução.

G. Critério de parada

No que diz respeito ao critério de parada, duas estratégias foram aplicadas. O algoritmo é encerrado quando o número máximo de iterações $k = k_{max}$ for atingido ou quando uma solução ótima é encontrada, isto é, $f(S_k) = 0$. Além disso, em termos de modições gerais que não impactam o desenvolvimento do método, mas facilitam o entendimento e *debug* do programa foram realizadas: exibição do número de iterações, solução corrente, valor da função objetivo, solução incumbente e seu número de ataques. No que diz respeito a programação, ela foi realizada na plataforma MATLAB, na qual o código consiste em uma função a ser chamada pelo usuário. Cada etapa principal foi implementada na forma de função resultando em um código modular de compreensão simples.

IV. Resultados e Dicussão

O primeiro experimento consiste em avaliar a evolução do algoritmo de Busca Tabu com duas implementações distintas. A primeira foi uma versão simplificada vista em aula. A segunda, desenvolvida neste artigo, consiste em uma versão com critério de diversificação e vizinhança até encontrar um vizinho melhor. A Tabela I mostra as iterações do algoritmo para $N = 21$, já a Tabela II para $N = 70$. Nota-se que para ambas as soluções iniciais a implementação do artigo encontrou a resposta em menos iterações. Entretanto, é importante mencionar que isso não se trata de um comportamento geral da segunda implementação, mas sim um comportamento pontual. Portanto, pode-se concluir que em algumas circunstâncias a segunda implementação encontra uma resposta mais rapidamente do que a primeira. Além disso, observa-se que a

diferença do número de iterações aumenta na medida em que mais rainhas são consideradas.

Tabela I
ITERAÇÕES DO ALGORITMO COM $N = 21$ RAINHAS.

$\mathbf{k}$	$\mathbf{f_1(S_k)}$	$\mathbf{f_1(S_{inc})}$	$\mathbf{f_2(S_k)}$	$\mathbf{f_2(S_{inc})}$
0	8	8	9	9
1	5	5	8	8
⋮	⋮	⋮	⋮	⋮
7	1	1	2	2
8	1	1	0	0
9	1	1	–	–
10	0	0		

Tabela II
ITERAÇÕES DO ALGORITMO COM $N = 70$ RAINHAS.

$\mathbf{k}$	$\mathbf{f_1(S_k)}$	$\mathbf{f_1(S_{inc})}$	$\mathbf{f_2(S_k)}$	$\mathbf{f_2(S_{inc})}$
0	33	33	40	40
1	29	29	33	33
⋮	⋮	⋮	⋮	⋮
15	1	1	1	1
16	1	1	0	0
23	1	1	–	–
24	0	0		

O segundo experimento teve como meta demonstrar o número médio de iterações, com inicialização aleatória, para 100 repetições do algoritmo conforme a Tabela III. O primeiro ponto a ser mencionado é que o intervalo de valores de N foi reduzido, pois na medida em que mais rainhas são consideradas maior é o tempo para encontrar uma solução ótima. O segundo ponto é que a primeira implementação levou menos iterações, na média, para chegar na resposta. Isso é explicado pelo método de geração de vizinhos, que no segundo caso para assim que um vizinho melhor (com menor número de ataques entre as rainhas) é encontrado. Já no primeiro são realizadas todas as permutações e então o melhor vizinho é selecionado.

Tabela III
NÚMERO MÉDIO DE ITERAÇÕES PARA 100 REPETIÇÕES.

$\mathbf{N}$	$\mathbf{f_1}$	$\mathbf{f_2}$
10	8	8
20	9	10
30	11	12
40	13	14

Para melhor ilustrar este comportamento foi elaborada a Figura 5 que exibe o valor da função objetivo f_2 (aquela que contabiliza mais ataques que f_1) ao longo das iterações. Pode-se observar que partindo da mesma solução inicial, quanto melhor for explorada a vizinhança $T_v = 1$, menor deve ser o número de iterações em comparação com $T_v = 2$. No caso desta figura, o primeiro método levou apenas 40 iterações ao passo que o segundo levou 103 iterações. Ademais, nota-se que no segundo caso a função objetivo ficou estagnada em maior quantidade de vezes fazendo uso do critério de diversificação em diversas ocasiões. Nesse sentido, o número de iterações em que a função objetivo apresenta o mesmo valor de forma consecutiva está diretamente associado ao parâmetro n_{est}. Ele delimita quando a solução corrente está estagnada e, portanto, quando a diversificação deve ser utilizada.

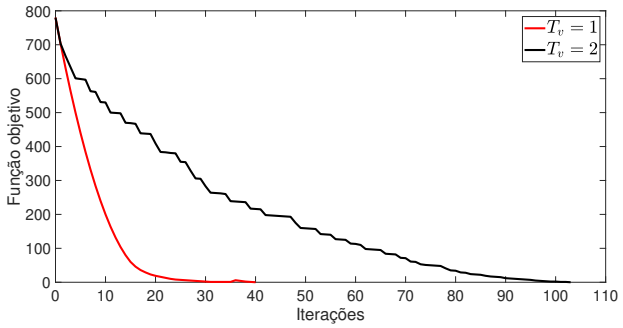

Figura 5. Influência da vizinhança.

Outra comparação que também foi realizada diz respeito à influência dos parâmetros de diversificação no desempenho do algoritmo de Busca Tabu. A Figura 6 mostra a evolução de f_2 para 100 soluções iniciais determinadas de modo aleatório e com $N = 30$. O parâmetro n_{est} foi definido como sendo igual a 5 e $n = 3$ (número de movimentos executados na diversificação). Neste cenário foi constatado que o número de ataques varia entre 9 e 35 iterações, sendo identificadas em vermelho tais situações. Outro comportamento presente nesta figura são os "saltos" na função objetivo que ocorrem no intervalo de 15 a 30 iterações resultantes da aplicação do critério de diversificação. Ademais, nota-se que a maioria das soluções permanece estagnada durante algumas iterações quando a solução corrente se aproxima de uma resposta ($f_2(S_k) = 1$).

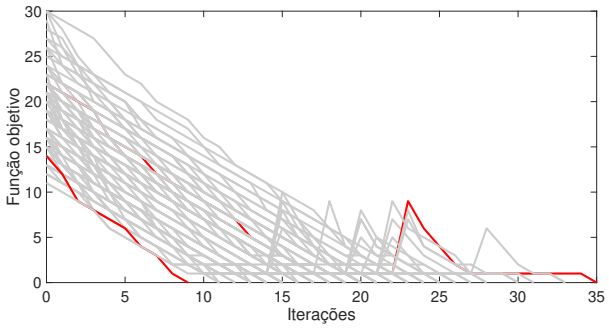

Figura 6. Influência do critério de diversificação com $n = 3$.

No intuito de melhor caracterizar os incrementos da função objetivo, repetiu-se o experimento anterior com os mesmos parâmetros à exceção de n que agora é definido como sendo igual a 10. O resultado é registrado na Figura 7. O primeiro item a ser destacado é que número de ataques varia entre 10 e 38 iterações. O segundo, e mais relevante, é que ao passo que o intervalo de ocorrência dos saltos permeneceu praticamente o mesmo, o grau de amplitude sofreu um aumento considerável da ordem 9 unidades, no caso anterior, para cerca de 20 unidades. Isso é explicado pois com $n = 10$ ocorrem mais alterações na solução corrente durante a aplicação da diversificação, o que acaba provocando um aumento no número de ataques. Por fim, é possível verificar que após a ocorrência do saltos, na maioria das vezes, a solução se encaminha para uma região "promissora" (com poucos ataques) sem a necessidade de repetição do critério de diversificação.

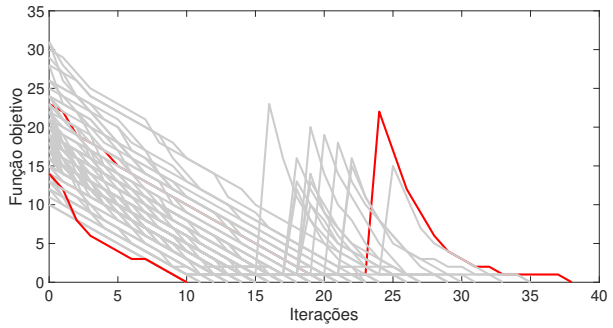

Figura 7. Influência do critério de diversificação com $n = 10$.

O último experimento realizado foi alterar o parâmetro n_{est} que determina quantas iterações a solução corrente deve permanecer inalterada para que o critério de diversificação ocorra. Assim, repetiu-se o experimento realizado para construir a Figura 6 considerando $n_{est} = 6$. O resultado é demonstrado na Figura 8. Em primeiro lugar, é possível constatar que o número de ataques varia entre 9 e 35 iterações. Em segundo lugar, vale destacar que o nível de amplitude após os saltos permanece praticamente o mesmo, já o que muda é o intervalo de ocorrência que agora é de 18 a 30. A justificativa é o incremento do parâmetro n_{est} cuja consequência é que a mais iterações devem ocorrer para que o critério de diversificação seja atendido.

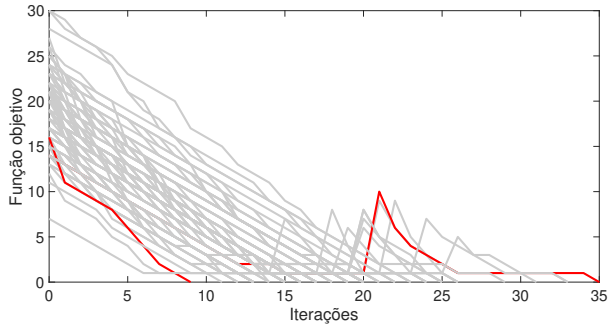

Figura 8. Influência do critério de diversificação com $n_{est} = 6$.

V. Conclusão

Em termos de características gerais observadas ao longo do desenvolvimento do trabalho, é possível mencionar que:

a) foram estudadas duas maneiras de computar o número de ataques. A diferença entre elas reside na forma de contabilizar os conflitos nos grupos de rainhas. A primeira simplismente é igual ao número indivíduos do grupo menos uma unidade. Já a segunda realiza uma combinação dos indivíduos do grupo para determinar seu número de ataques. O que vale ser destacado é que se o número de ataques é diferente, então no momento de classificar os vizinhos utilizar uma ou outra função objetivo faz total diferença na sequência de movimentos a ser realizada. Por exemplo, um vizinho com três ataques calculados com f_1 pode ter seis ataques se calculado com f_2. Assim, o que é o melhor para f_1, em geral, não é para f_2, o que implica em movimentos subsequentes distintos;

b) foram implementadas duas estratégias para calcular a vizinhança. Sob certas circunstâncias, ou seja, para algumas soluções iniciais o algoritmo é capaz de encontrar a resposta mais rapidamente utilizando a vizinhança até o encontrar uma solução melhor do que na vizinhança completa. Por outro lado, no caso geral, quanto melhor é explorada a vizinhança menor é número de iterações da Busca Tabu. Essa diferença tende a aumentar na medida em que mais rainhas são consideradas;

c) a ideia de implementar o critério de diversificação a partir da matriz trigular inferior da matriz de tabus se mostrou simples e efetiva. Os movimentos não executados são facilmente detectados verificando onde ocorrem zeros. A partir disso, realiza-se um sorteio entre esses movimentos para que sejam aplicados na solução corrente;

d) houve variabilidade de desempenho do algoritmo conforme os parâmetros do critério de diversificação. Quanto maior o valor de n_{est}, mais iterações devem ocorrer com a função objetivo estagnada, ou seja, a diversificação é aplicada com menor frequência. Graficamente, isso acontece através de um deslocamento dos saltos da função objetivo para a direita (atraso). Além disso, quanto maior o valor de n, maior é a amplitude dos saltos, já que ocorrem mais modificações na solução corrente durante a diversificação. Na maioria dos cenários avaliados, o critério de diversificação ocorre apenas uma vez e quando a solução está próximo da resposta.

A partir destes elementos, como sugestões para trabalhos futuros pode-se mencionar: propor outra técnica de geração de vizinhos que não seja a permuta, por exemplo, avaliar apenas os movimentos que envolvam rainhas sob ataque; implementar mecanismo de intensificação visando explorar as melhores regiões já visitadas; aplicar um critério de diversificação combinando características da solução incumbente com regiões ainda não exploradas (alguma mistura dos operadores de mutação dos algoritmos genéticos para problemas de permutação); desenvolver um mecanismo de curto prazo com múltiplas listas tabu como mencionado em [6]; investigar formas de determinar soluções únicas no problema das N-rainhas; desenvolver uma *interface* gráfica para auxiliar o usuário na execução e entendimento de cada etapa do algoritmo; realizar uma análise estatística para melhor caracterizar a eficiência dos mecanismos e modificações implementadas e, por último, comparar o desempenho da Busca Tabu com demais métodos clássicos de otimização heurística: algoritmo genético, recozimento simulado, colônia de formigas, enxame de abelhas, entre outros.

Agradecimentos

Os autor agradece ao professor Sérgio Luis Haffner pela oportunidade de realização deste trabalho uma vez que ele proporcionou o estudo não apenas do algoritmo Busca Tabu, mas também do algoritmo de *backtracking*, busca local iterativa e da heurística de conflitos mínimos.

Referências

[1] H. M. F. de Souza, "Algoritmo eficiente para validação de soluções para o problema de N-rainhas," Monografia (Ciência da computação), Universidade Federal de Alagoas, Arapiraca, 2019.

[2] I. Martinjak and M. Golub, "Comparison of Heuristic Algorithms for the N-Queen Problem," in 29*th International Conference on Information Technology Interfaces*, 2007, pp. 759–764.
[3] C. Moreira Oliveira and A. Pozo, "Resolução para o problema n-rainhas utilizando ACO," *Encontro Nacional de Inteligência Artificial e Computacional*, vol. 1, pp. 353–358, 01 2014.
[4] S. Gudal, V. Baugh, and S. Allehaibi, "N-Queens Solving Algorithm by Sets and Backtracking," in *SoutheastCon*, 2016, pp. 1–8.
[5] S. Sharma and V. Jain, "Solving N-Queen Problem by Genetic Algorithm using Novel Mutation Operator," *IOP Conference Series: Materials Science and Engineering*, vol. 1116, no. 1, p. 012195, apr 2021.
[6] E.-G. Talbi, *Metaheuristics - From Design to Implementation*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2009.
[7] F. Glover, "Tabu Search—Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.
[8] M. Laguna, "A Guide to Implementing Tabu Search," *Investigación Operativa*, vol. 4, no. 1, pp. 5–25, 1994.