Solver Close or Far

André Rocha[1,2,3][up201706462] and Manuel Coutinho[1,2,3][up201704211]

[1] Faculdade de Engenharia da Universidade do Porto - PLOG
[2] Turma 3MIEIC06
[3] Grupo Close_Or_Far_5

Resumo. Na Unidade Curricular de Programação em Lógica foi-nos proposta a escrita de um artigo relativamente ao desenvolvimento de um *Solver* para os Puzzles desenvolvidos pelo professor Erich Friedman [1] de nome *Close or Far* em Prolog usando Programação em Lógica por Restrições. Serão apresentadas duas formas de restringir o problema, seguido de duas estratégias diferentes para a geração de tabuleiros de um dado tamanho, sendo que uma destas garante unicidade da solução do problema.

1 Introdução

O presente artigo irá apresentar um programa escrito em SICSTUS Prolog com o intuito de resolver Puzzles *Close or Far* [2]. Apesar de uma modéstia quantia de puzzles do tipo existirem, estes possuem todos um tamanho reduzido, pelo que não providenciavam a complexidade necessária para testar efetivamente a qualidade de um solver. Deste modo, foram também criados dois geradores de Puzzles, distinguidos pelo facto de que apenas um deles garante uma solução única ao puzzle, algo que aumenta significativamente a complexidade da solução.

Este artigo irá começar por fazer uma descrição aprofundada sobre as regras que cada puzzle segue, para poder proceder a uma explicitação da abordagem seguida para a resolução do problema de decisão. Serão apresentadas todas as variáveis de decisão presentes bem como os seus respetivos domínios, como são restringidos os seus valores, e a estratégia de pesquisa tomada.

Será ainda explanado o que é usado para a visualização do problema e os respetivos resultados obtidos, através da demonstração dos resultados obtidos com diferentes heurísticas de escolha. O artigo irá terminar com uma sucinta conclusão de todo o desenvolvimento, evidenciando alguns aspetos que poderiam ser melhorados futuramente.

2 Descrição do Problema

Close or Far é um problema de decisão. Cada puzzle segue uma série de regras específicas que serão agora explicadas:

- Cada casa poderá estar vazia, ter um C (*Close*) ou um F(*Far*).

– Cada linha e coluna deve possuir exatamente 2 instâncias de cada letra.
– Cada linha e coluna deve garantir que a distância entre os C's presentes nela seja menor que a distância entre os F's.

Quando todas estas condições se encontram satisfeitas, considera-se que o puzzle está completo.

C	C	F		F	
C	F	C			F
F	C	C	F		
		F	C	C	F
	F		C	F	C
F			F	C	C

Fig. 1. Exemplo de um puzzle terminado de Close or Far

3 Abordagem

3.1 Variáveis de Decisão

Um tabuleiro de *Close or Far* de tamanho NxN é representado através de uma lista composta por N listas com N elementos cada uma com um domínio entre 0 e 2 representando o 0 a casa vazia, 1 o carácter C (*Close*) e 2 o F (*Far*)

Exemplificando, um tabuleiro completo 6x6 terá, em cada uma das suas 6 listas, 2 peças do tipo 1, 2 peças do tipo 1 e as restantes (duas) do tipo 0.

```
board([[1, 1, 2, 0, 2, 0],
       [1, 2, 1, 0, 0, 2],
       [2, 1, 1, 2, 0, 0],
       [0, 0, 2, 1, 1, 2],
       [0, 2, 0, 1, 2, 1],
       [2, 0, 0, 2, 1, 1]]).
```

Fig. 2. Representação em Prolog do tabuleiro da Fig. 1

3.2 Restrições

Para a resolução deste problema, apenas foram usadas restrições rígidas, obrigando a que a resposta resultante fosse, efetivamente, a resolução do puzzle em questão.

Assim, duas abordagens foram testadas: uma que se trata de uma tradução quase direta das regras específicas do puzzle; e outra que passa pela implementação de um autómato (devido a sugestão do professor com base na experiência de alunos de anos anteriores) que tornou os tempos de execução consideravelmente melhores quando aplicadas apenas as heurísticas definidas por omissão (explanado no Capítulo 3.3), o que não reflete os resultados finais.

De notar que as restrições foram aplicadas a cada linha recorrendo ao predicado *maplist*, seguindo-se de uma rotação da matriz que representa o tabuleiro para aplicação das mesmas às colunas (agora linhas).

Abordagem de tradução direta Como explicado, esta abordagem resume-se à aplicação das regras traduzidas para prolog: dois elementos do tipo 1, dois elementos do tipo 2 e os restantes (N - 4) do tipo 0; distância entre 1's < distância entre 2's.

```
restrict(Line):-
    domain(Line, 0, 2),
    length(Line, LineLength),
    NumZeroes is LineLength - 4,
    global_cardinality(Line, [0-NumZeroes, 1-2, 2-2]),
    element(ElemC1, Line, 1),
    element(ElemC2, Line, 1),
    ElemC1 #< ElemC2,
    element(ElemF1, Line, 2),
    element(ElemF2, Line, 2),
    ElemF1 #< ElemF2,
    DistC #= abs(ElemC1 - ElemC2),
    DistF #= abs(ElemF1 - ElemF2),
    DistC #< DistF.
```

Fig. 3. Implementação da abordagem *naive*

Abordagem com autómato Para melhor compreensão desta abordagem, faremos acompanhar esta secção, não com o predicado correspondente, mas com o esquema que representa o DFA implementado. A ideia geral é um máquina que obrigue a que haja exatamente dois 1's e dois 2's, contando o número de elementos entre cada par e guardando-os em acumuladores distintos C e F, respetivamente.

Como podemos reparar ao longo das letras (a, b, c) o número de 2's aumenta e ao longo dos números contamos o número de 1's. Também podemos constatar que nos estados intermédios (entre dois 1's, dois 2's ou ambos) há um incremento do acumulador correspondente (por exemplo em a2, quando acrescentamos 0's, como já foi lido um 1, a distância entre estes aumenta).

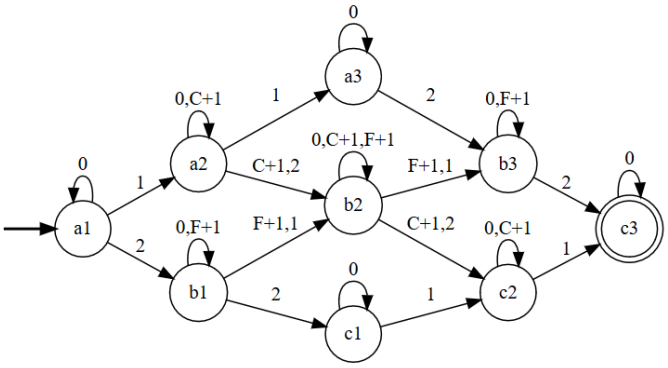

Fig. 4. Autómato Implementado

3.3 Estratégia de Pesquisa

É ainda relevante mencionar a Estratégia de Pesquisa tomada. É de notar que a escolha das heurísticas mais relevantes neste ponto foram um dos principais fatores na redução do tempo de pesquisa em puzzles de maior tamanho, conseguindo reduções astronómicas face aos resultados com as heurísticas presentes por omissão. É importante mencionar que qualquer referência à incapacidade de um puzzle ser terminado em tempo útil implica que este não foi capaz de encontrar uma solução num tempo inferior a 60s (cerca do dobro do tempo para ser encontrado uma solução num puzzle de tamanho 100x100 com as heurísticas por omissão).

Certas heurísticas provaram-se como sendo completamente inadequadas ao problema, como, por exemplo, *min* e *max*, ambas estratégias de Ordenação. Uma vez que o valor da variável não tem qualquer significado inerente ao problema, estas escolhiam um caminho substancialmente mais lento, pelo que não tinham a capacidade de terminar em tempo útil puzzles extremamente simples, de tamanhos menores a 20x20.

Após a análise dos Resultados posteriormente explicitados no Capítulo 7, foi possível de determinar as heurísticas com os resultados mais promissores para a pesquisa de uma solução:

- *ffc* - Estratégia de Ordenação de Variáveis que escolhe a variável de menor domínio, determinando a que se encontra com o maior número de restrições como meio de desempate. Caso isto não seja suficiente para ocorrer um desempate, é escolhida a variável mais à esquerda. Esta restrição apresenta-se como ideal para a resolução dos puzzles do tipo *Close or Far* uma vez que este tipo de táctica conseguirá rapidamente eliminar as posições do tabuleiro que já não poderão receber peças do tipo 1 e 2.
- *middle* - Estratégia de Seleção de Valores recorrendo a uma escolha binária entre se a variável em causa é a média do domínio ou se é a diferente da média do domínio.

A utilização desta combinação de heurísticas relevou melhorias significativas em ambas as abordagens desenvolvidas para resolver o problema. Foi ainda possível de concluir que a primeira abordagem acaba por conseguir melhores resultados do que a Abordagem com autómato, revelando-se como sendo um alternativa substancialmente mais rápida comparativamente a uma abordagem inicialmente pensada como sendo mais avançada e, consequentemente, mais eficiente. É pertinente referir que a abordagem que nos parecia mais *naive*, antes da introdução de heurísticas, obtinha resultados tão lentos, que a incapacitavam de resolver um puzzle de tamanho 10x10 em menos de 10 minutos; após a introdução das heurísticas, esta conseguiu terminar um puzzle de tamanho 100x100 com sucesso em cerca de 1.22 segundos. Para demonstração do que foi explanado anteriormente, é apresentado um gráfico de comparação entre ambas as abordagens usando as heurísticas *ffc* e *middle*.

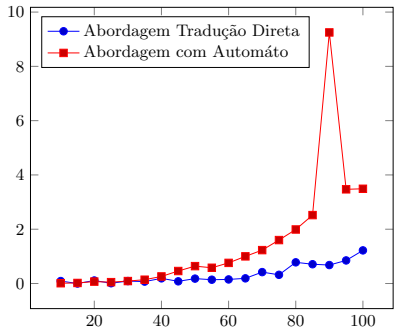

Fig. 5. Comparação da pesquisa da combinação de *middle* com *ff*, *ffc* e *max_regret*

4 Visualização da Solução

O predicado de visualização do tabuleiro foi reaproveitado na sua maioria do trabalho anterior, assim, através a chamada de *display_board(+Board)*, o puzzle resolvido é escrito recursivamente, linha a linha, no terminal, assemelhando-se à sua representação normal.

De notar que, para que o tabuleiro fique formatado como se pretende, é aconselhado o uso de um ambiente Linux ou uma fonte *monospace*, para garantir o correto espaçamento entre as células.

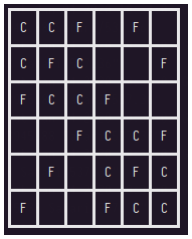

Fig. 6. Visualização da Solução apresentada na Figura 1

5 Geração

Para que o próximo capítulo tivesse alguma significância, foi necessária a criação de um predicado que gerasse puzzles de tamanhos maiores do que os inicialmente encontrados no site. Assim, foram criados dois geradores: um que gera puzzles com solução única e outro, mais rápido, que consegue gerar puzzles de maior tamanho.

5.1 Geração de Puzzles Geral

Começa-se por resolver uma matriz vazia do tamanho desejado com o *solver* previamente criado, seguindo-se da seleção das *hints* para o puzzle final.

Este processo seletivo foi desenvolvido a partir de uma generalização dos puzzles que se encontram na página original: todos estes tinham apenas um elemento preenchido por linha e coluna (estando todos os outros vazios), por isso, utilizou-se uma versão simplificada do Problema das Rainhas[4][5] onde, em vez das restrições diagonais, se colocou uma restrição a impedir que o elemento escolhido fosse um espaço vazio (elemento 0).

5.2 Geração de Puzzle com Solução Única

Esta geração resulta apenas da aplicação de mais uma condição ao predicado anterior.

Várias estratégias foram experimentadas sem sucesso para tornar a geração num mecanismo *constraint and generate* ao invés de um *generate and test*.

Ficamos, assim, com duas abordagens possíveis, ambas elas com resultados medíocres para puzzles de tamanhos superiores a 7 por 7:

- ***Findall***: Recorrer a este predicado para encontrar todas as soluções de um puzzle gerado e garantir que o retorno é 1.
- **Autómato**: Através de garantir a negação de um predicado que encontra uma solução diferente da original (usando um autómato muito simples para garantir que são efetivamente diferentes, pois existe pelo menos um elemento diferente).

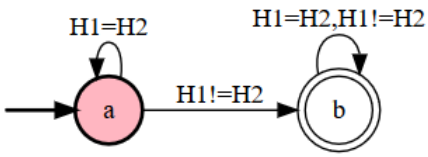

Fig. 7. Autómato Implementado para Detetar se são Diferentes

6 Resultados

Foi desenvolvido um pequeno módulo com a intenção testar todas as combinações de 1 predicado de Ordenação de Variáveis com 1 predicado de Seleção de Valores. Cada uma destas combinações resolve os tabuleiros de tamanho 10x10 até 100x100 (inclusive), gerados pelos predicados do capítulo anterior, com um incremento de 5 unidades a ambos os lados por iteração. Os resultados obtidos são apresentados no Anexo A, sendo que é relevante de mencionar que nem todas as combinações de predicados são apresentadas, uma vez que diversas combinações não conseguem terminar em tempo útil, levando a uma situação de *timeout*.

De notar que todos os resultados apresentados neste capítulo foram capturados usando a Abordagem de Restrição com autómato, devido à sua maior flexibilidade em receber diferentes heurísticas e obter resultados em tempo útil, além de que havia uma correlação do melhoramento do tempo desta com o da primeira abordagem.

Com base nos resultados, é possível afirmar que os predicados *min*, *max* e *anti_first_fail* de Ordenação não são adequados ao problema, uma vez que nenhuma das suas possíveis combinações com predicados de Seleção conseguiram gerar a sequência de tabuleiros em tempo útil. Para além disso, o predicado *median* para Seleção de Valores também apresentou resultados semelhantes, pelo que foi também considerado inadequado.

No entanto, certas estratégias demonstraram resultados extremamente promissores para a resolução de tabuleiros de *Close or Far* . Destes, é de destacar a estratégia de Seleção de Valores *middle* que conseguiu os melhores e mais estáveis resultados em todas as combinações em que teve presente. A sua combinação com a estratégia de Ordenação *ffc* conseguiu os resultados mais expressivos, obtendo reduções de cerca de 90% face à pesquisa com os valores por omissão (redução ainda mais significativa na outra abordagem). No entanto, estratégias como *max_regret* e *ff* também foram bem sucedidas na sua combinação com *middle*. É de seguida, portanto, apresentado um gráfico que compara estas 3 combinações com resultados mais promissores.

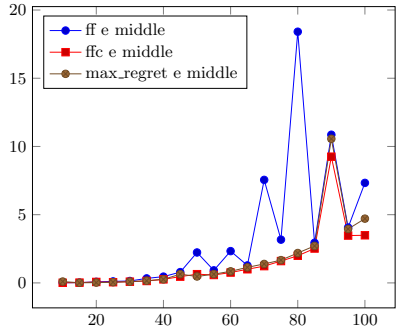

Fig. 8. Comparação da pesquisa da combinação de *middle* com *ff*, *ffc* e *max_regret*

O gráfico apresentado esclarece precisamente o que foi explicitado, demonstrando como a combinação da estratégia *ffc* com *middle* pode de facto obter os melhores resultados, no entanto, a estratégia *max_regret* obtém valores extremamente semelhantes. A combinação com a estratégia *ff* gera valores que, apesar de apresentarem uma significativa maior irregularidade em comparação com os outros, não deixa de ser uma das melhores estratégias de pesquisa encontradas.

7 Conclusão

Deste projeto foi possível possível retirar a importância que uma heurística pode ter na resolução de um problema. A capacidade de escalar de uma solução quando dadas as melhores heurísticas para resolver um problema é substancialmente mais elevada. A solução desenvolvida para o referido problema demonstra exatamente isso: como uma abordagem inicialmente descartada pela sua incapacidade em resolver puzzles de complexidade baixa em tempo útil, acabou por se revelar sendo ainda melhor que uma abordagem muito mais complexa que, quando apresentada com as heurísticas por omissão, obtinha resultados consideravelmente melhores, conseguindo resolver puzzles com alguma complexidade instantaneamente.

Há, contudo, algumas partes que poderiam ser melhoradas nomeadamente a geração de puzzles com solução única. Apesar da mudança do uso do predicado *findall* para *two_solutions* ter demonstrado um melhoramento dos resultados, estes ainda se encontram muito aquém do desempenho que gostaríamos que apresentassem. Para tal teríamos de passar de uma abordagem *generate and test* para uma mais *constraint and generate*.

Poderíamos ainda melhorar marginalmente o autómato encontrado no predicado *restrict2* alocando estados dependendo do tamanho da lista que contassem implicitamente o número de 0's na linha ou coluna, restringindo-o assim também em tempo de leitura, contudo consideramos que o facto de já estar implicitamente restringido faz com que esta não fosse trazer melhoramentos temporais significativos que pudessem competir com o predicado *restrict*.

Bibliografia

1. Erich's Place, https://www2.stetson.edu/ efriedma/. Acedido pela última vez a 1 Jan 2020.
2. Close or Far, https://www2.stetson.edu/ efriedma/puzzle/closefar/. Acedido pela última vez a 1 Jan 2020.
3. Enumeration predicates,https://sicstus.sics.se/sicstus/docs/4.2.0/html/sicstus/Enumeration-Predicates.html Acedido pela última vez a 5 Jan 2020.
4. Eight queens puzzle, https://en.wikipedia.org/wiki/Eight_queens_puzzle Acedido pela última vez a 5 Jan 2020.
5. Solving N-queens with Prolog, https://www.metalevel.at/queens/. Acedido pela última vez a 5 Jan 2020.

A Gráficos com diferentes opções de *Labeling*

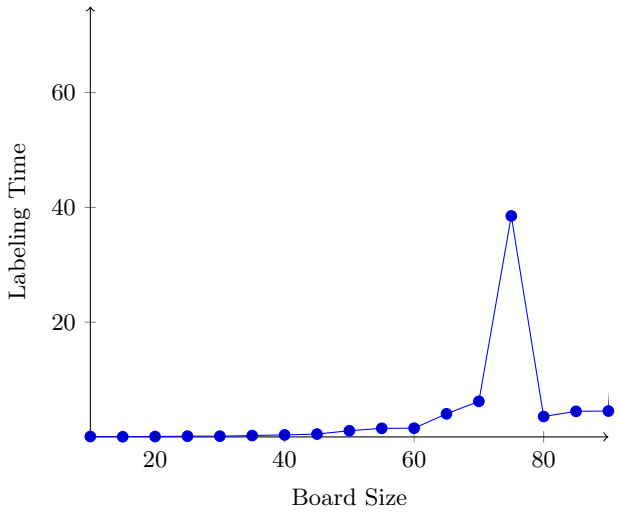

Fig. 9. *Labeling* usando as opções ff e bisect

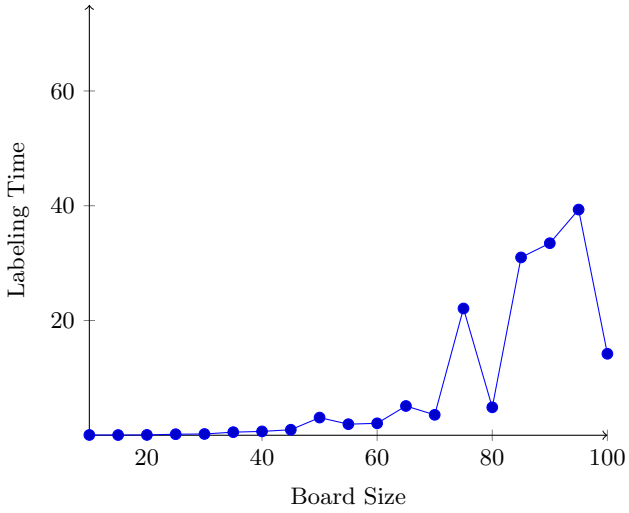

Fig. 10. *Labeling* usando as opções ff e enum

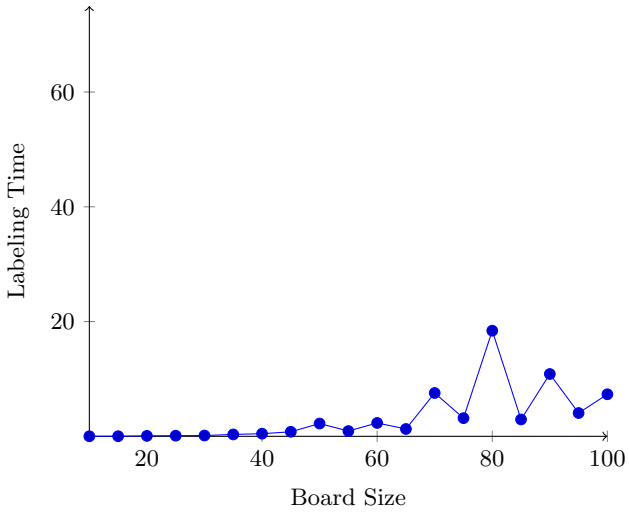

Fig. 11. *Labeling* usando as opções ff e middle

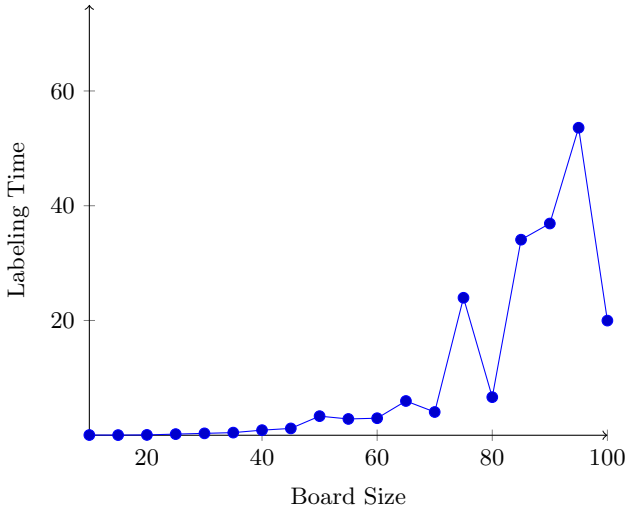

Fig. 12. *Labeling* usando as opções ffc e bisect

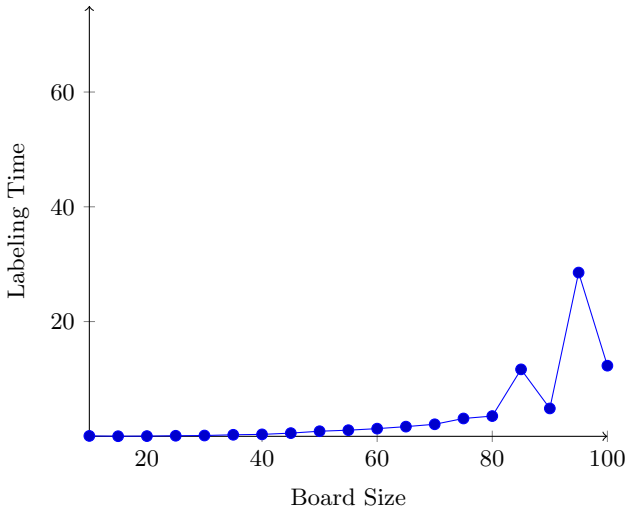

Fig. 13. *Labeling* usando as opções ffc e enum

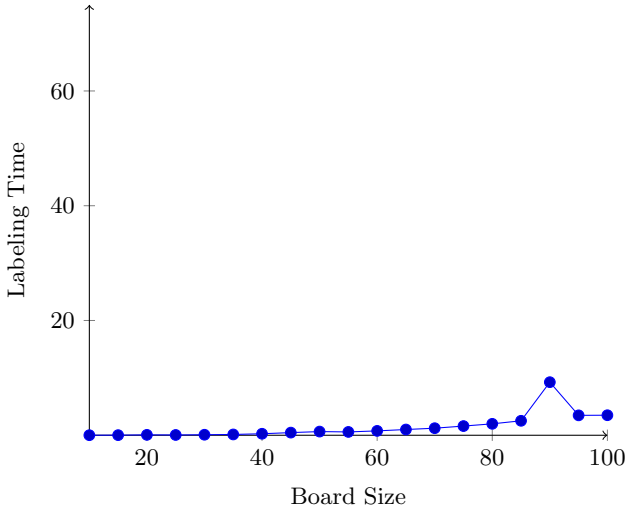

Fig. 14. *Labeling* usando as opções ffc e middle

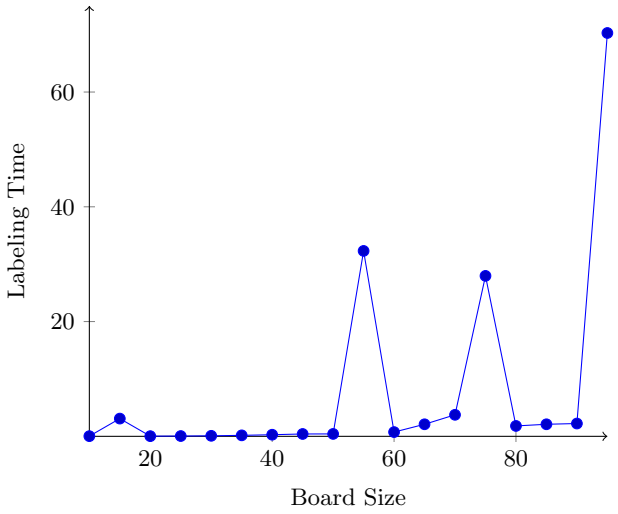

Fig. 15. *Labeling* usando as opções leftmost e bisect

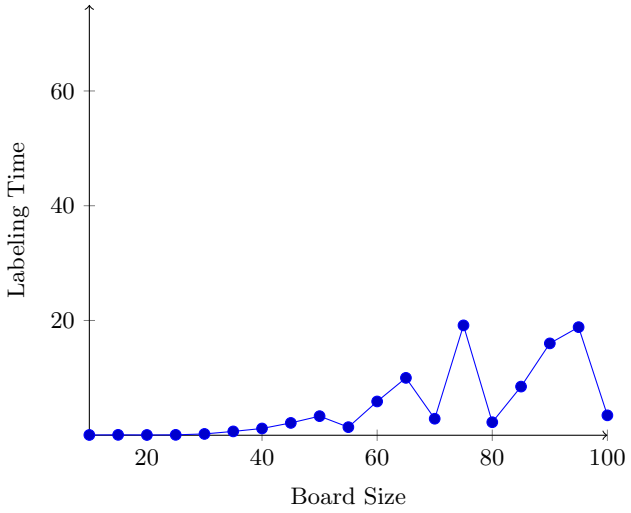

Fig. 16. *Labeling* usando as opções leftmost e enum

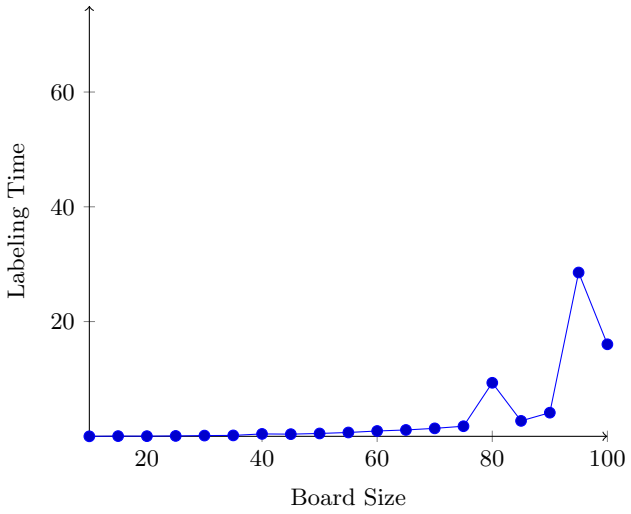

Fig. 17. *Labeling* usando as opções max_regret e bisect

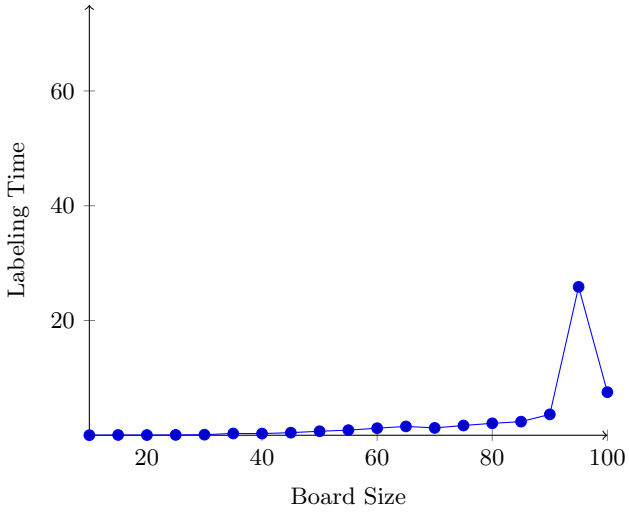

Fig. 18. *Labeling* usando as opções max_regret e enum

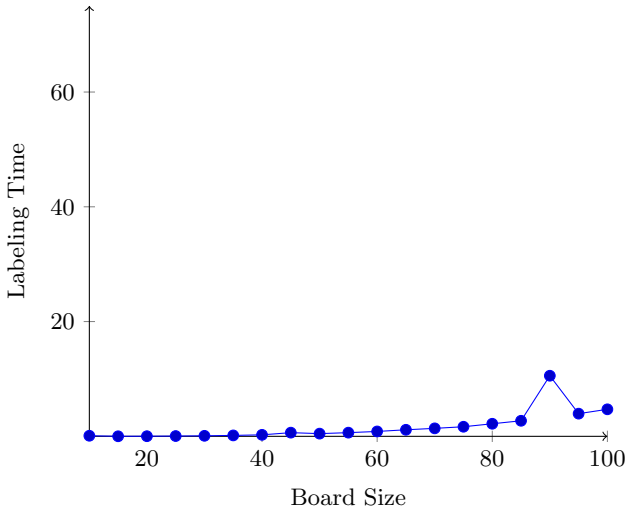

Fig. 19. *Labeling* usando as opções max_regret e middle

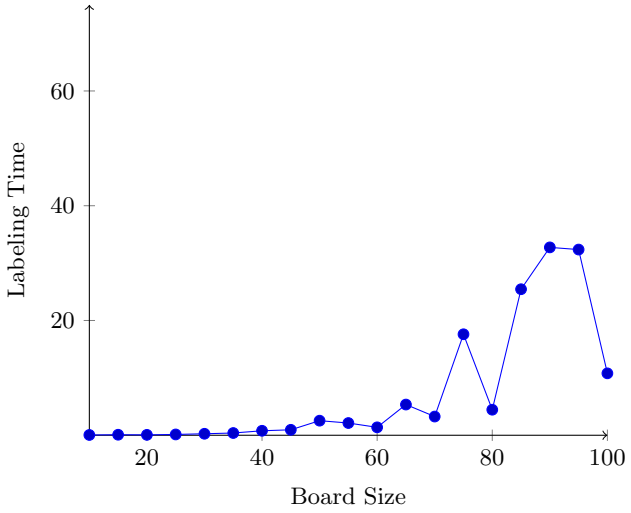

Fig. 20. *Labeling* usando as opções ocurrence e bisect

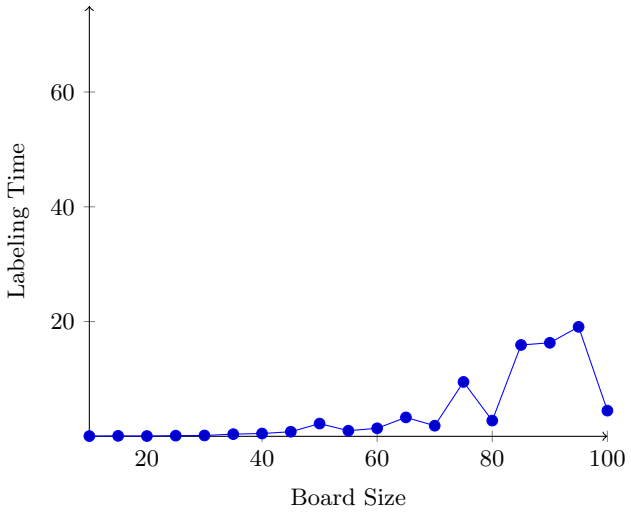

Fig. 21. *Labeling* usando as opções ocurrence e enum