Discente: João Vítor Antunes Ribeiro **NUSP:** 9744910

Relatório Técnico – Projeto "*Backtracking*"

- **Requisitos:** Os requisitos de *software* e *hardware* utilizados para o desenvolvimento do projeto foram, respectivamente: Sistema Operacional Windows 10 Pro x64 (2015), 4GB de memória RAM e processador Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz 2.60GHz.

- **Interface gráfica:** O projeto foi desenvolvido utilizando o *Eclipse Java EE IDE for Web Developers*, versão 4.5.2. (2016). Para implementar a interface gráfica, foi utilizada a extensão *WindowBuilder*, disponível especialmente para o *Eclipse*. A **Figura 1** ilustra a interface gráfica da ferramenta *Sudoku 3x3 Resolver*. O "Menu de arquivos" possui as opções de abrir um arquivo de problemas (atualiza a área de "Entrada dos problemas"), salvar ou limpar o conteúdo atual da área de "Entrada dos problemas", contida na aba **Input** (em destaque na **Figura 1**). O menu à direita da janela da interface contém as opções de "Seleção do algoritmo", que pode ser "BT" (*Backtracking* simples), "BT+MRV" (*Backtracking* com seleção de Mínimos Valores Remanescentes), "BT+FC" (*Backtracking* com heurística de Verificação Adiante (*Forward Checking*)) e "BT+MRV+FC" (Backtracking com seleção de MVR e heurística de FC).

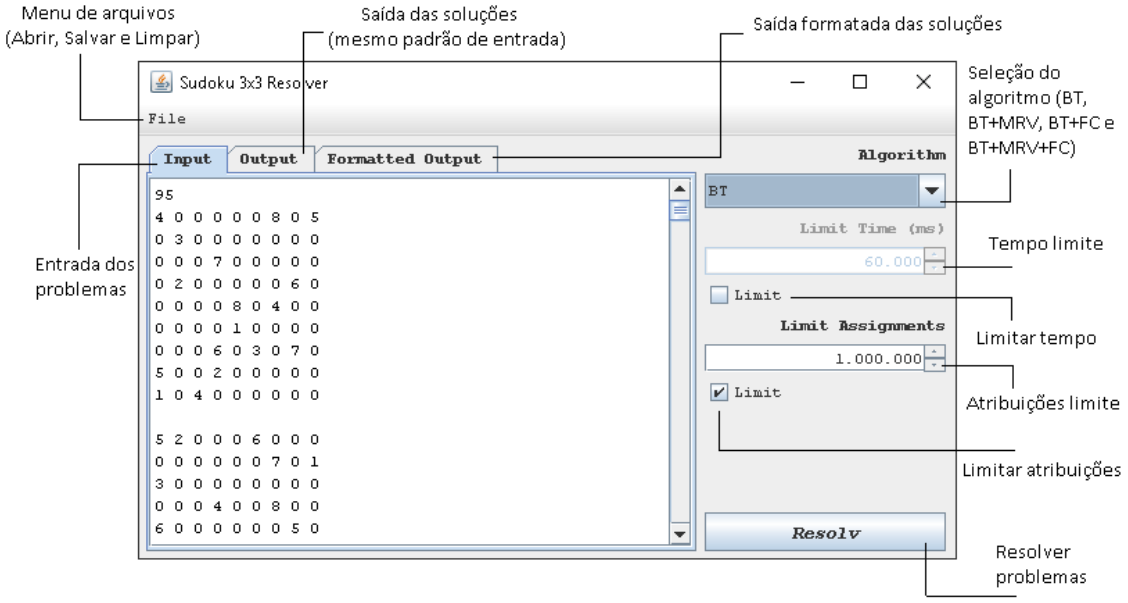

Figura 1: Interface gráfica da ferramenta *Sudoku 3x3 Resolver*.

- **Funcionamento:** Para processar corretamente um conjunto de problemas, a entrada deve estar formatada em texto puro, com a primeira linha indicando a quantidade de problemas a serem lidos, e as linhas seguintes especificando os problemas em si, com espaços simples separando as células, e cada linha separada por uma quebra de linha simples, assim como deve haver uma linha em branco entre todos os problemas, conforme ilustrado na **Figura 1**.

  Após a entrada do conjunto de problemas na área correspondente à aba **Input**, o usuário pode escolher um algoritmo de resolução do problema (por padrão BT). Também é possível limitar o processamento da resolução dos problemas, tanto por tempo quanto por quantidade de atribuições realizadas. Para isso, basta selecionar a caixa de seleção correspondente à cada tipo de limitação, bem como especificar o número limite. Para a limitação de tempo, o valor é dado em milissegundos (variando de 1 à 2.147.483.647). Já a limitação de número de atribuições corresponde à quantidade máxima de chamadas

recursivas à função de resolução de problemas Sudoku (variando de 1 à 2.147.483.647). É válido destacar que as limitações são independentes para cada problema a ser resolvido, e por isso não são cumulativas.

Para resolver os problemas da área de entrada de dados, o botão ***Resolv*** deve ser pressionado. Como a ferramenta não possui o artifício de *threads*, não é possível parar a execução (identificada quando o botão de processamento está pressionado) dos problemas uma vez que o botão de resolução é acionado. Para conjuntos grandes de problemas, o processo de resolução pode ser muito demorado, sendo aconselhável à utilizar os critérios de parada sempre que possível. Ao final do processamento, o usuário é redirecionado para a aba **Formatted Output**, ilustrada na **Figura 2**. Como é possível notar na **Figura 2**, de acordo com o critério de parada, alguns problemas podem não obter solução, sendo exibidos na cor vermelho, ao passo que os problemas resolvidos são exibidos na cor verde. Nessa aba, os problemas são identificados na ordem em que foram inseridos na entrada, exibindo se houve ou não solução, o tempo e a quantidade de atribuições realizadas, bem como quais critérios de parada foram excedidos, se for o caso. Para os problemas que não houve solução válida, a saída é o estado de processamento final da possível solução. Na aba **Output**, as soluções para os problemas são exibidas seguindo o mesmo padrão dos dados de entrada.

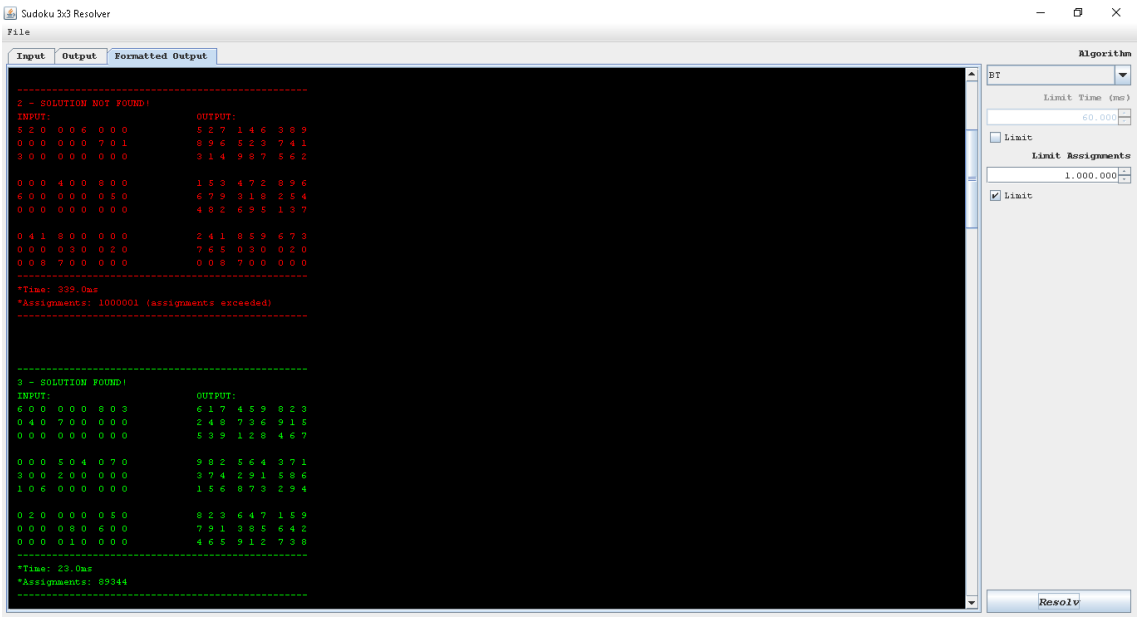

Figura 2: Exemplo de saída formatada.

- **Implementação:** A ferramenta foi desenvolvida utilizando a linguagem Java. Seu código fonte possui 3 *packages*: *gui* (contém a classe de interface gráfica *MainWindow.java*), *main* (contém a classe inicializadora da ferramenta *Main.java*) e *sudoku* (contém as classes principais para resolução dos problemas, *Resolver.java* e *Result,java*). Em ordem de execução, a classe *Main* instancia a classe *MainWindow*, que por sua vez faz a comunicação entre o usuário e a classe *Resolver*, que se comunica com a classe *Result*. A classe *Result* possui apenas os atributos de execução de um problema (a matriz do problema, uma matriz de resultado e uma matriz de valores remanescentes, além do tempo de execução, número de atribuições e uma *flag* indicando se o problema foi resolvido). A classe *Resolver*, por sua vez, é a principal da ferramenta, pois é através dela que os problemas são resolvidos logicamente. Assim, através de uma chamada da classe *MainWindow*, por meio do método *resolveAll()*, para cada problema do conjunto de entrada, é realizada uma chamada ao método *resolve()*, passando como parâmetro a instância de *Result* (a classe) daquele problema específico. O método *resolve()* é recursivo e, por meio de força bruta, garante a resolução dos problemas caso nenhum critério de parada esteja definido. Ele contém *flags* que podem selecionar a próxima célula a ser preenchida através de MVR ou não, de acordo com o método *isMVR().* Os critérios de parada são controlados pelo método *exceedStopCriteria()*, que atualiza o tempo e número de chamadas recursivas ao método *resolve()*. O método

isFC() é a flag responsável por verificar a utilização da heurística FC, e caso haja necessidade de backtrack, evita atribuições desnecessárias à resolução do problema. De forma geral, o algoritmo verifica a próxima célula a ser preenchida e testa todos os valores possíveis para ela; caso um valor seja válido e não cause inconsistência na solução (verificação realizada pelo método *allDiff()*), a próxima célula vazia é preenchida através de uma chamada recursiva, caso contrário, o *backtrack* é disparado e a última penúltima chamada testa o próximo número possível à célula anteriormente preenchida. Esse procedimento é realizado continuamente até que a solução seja encontrada. O método auxiliar *updateVR()* atualiza a quantidade de valores remanescentes para cada célula da matriz de resolução, utilizando o método *getNumberVR()* para realizar o cálculo; *getHoleLocation()* retorna a primeira célula vazia da matriz de resolução, se houver; *getHoleLocationMVR()* retorna a célula com menor número de valores remanescentes da matriz de resolução, se houver; e *needBacktrack()* verifica se pelo menos uma das células vazias da matriz de resolução não possui valores remanescentes.

- **Testes realizados:** A avaliação da ferramenta foi realizada com o objetivo de comparar as diferentes abordagens de resolução dos problemas, em tempo de execução e quantidade de atribuições. Para isso, foi utilizado um conjunto de 10 problemas Sudoku disponível na página da disciplina (arquivo *entrada10.txt*). Esse conjunto de testes foi processado uma única vez para cada abordagem diferente. A seguir, os tempos de execução para cada algoritmo são exibidos no **Quadro 1**, seguido do seu respectivo gráfico, ilustrado em **Gráfico 1**. Os dados sobre a quantidade de atribuições realizadas nos testes são exibidos em seguida, dispostos no **Quadro 2** e ilustrados no **Gráfico 2**. Ambos os gráficos são de colunas 100% empilhadas, para permitir melhor comparação visual entre os valores muito discrepantes ou distantes.

		Tempo (ms)			
		BT	BT+MRV	BT+FC	BT+MRV+FC
Problema	1	2503	11	25271	3
	2	893	130	13901	128
	3	21	29	1152	30
	4	107	1335	1836	1369
	5	31516	3317	287057	3447
	6	78	1360	1199	1790
	7	1762	5411	48750	6483
	8	1383	14	26249	14
	9	1173	3220	36452	3340
	10	2679	1507	47593	1538

Quadro 1: Tempos de execução dos testes realizados.

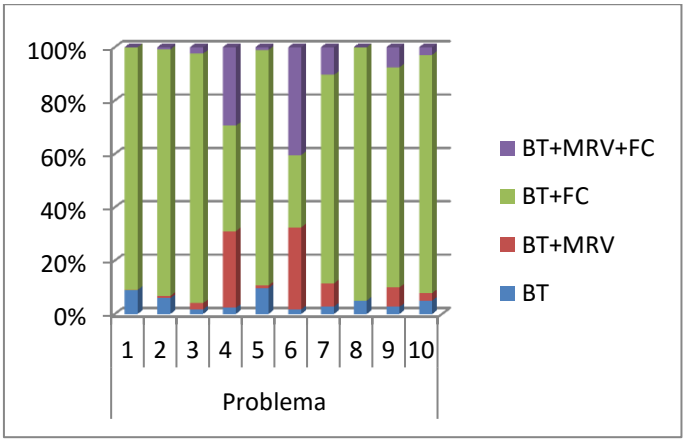

Gráfico 1: Tempos de execução dos testes realizados.

Atribuições

		BT	BT+MRV	BT+FC	BT+MRV+FC
Problema	1	9727397	315	1029095	297
	2	3252581	8726	842830	7614
	3	89344	1994	54326	1758
	4	392886	89057	94587	76717
	5	112256362	257225	14036928	226959
	6	291406	103338	67746	91990
	7	6996436	386773	2108921	331082
	8	5688443	1091	1230536	960
	9	4803036	232724	1547122	198893
	10	11471000	100218	2108778	86111

Quadro 2: Número de atribuições dos testes realizados.

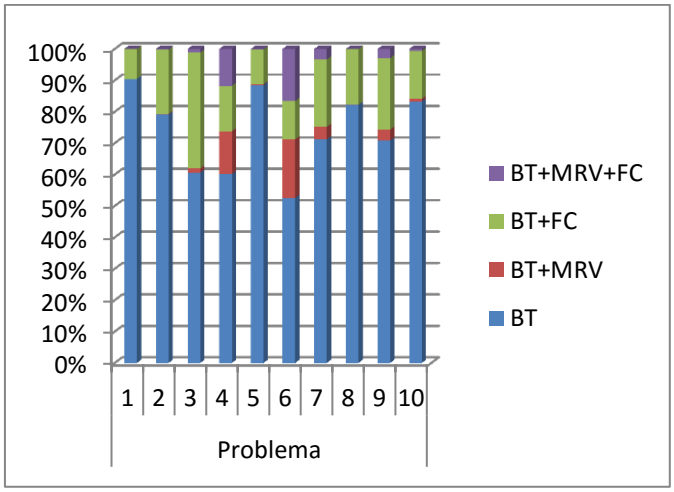

Gráfico 2: Número de atribuições dos testes realizados.

- **Conclusões:** Os gráficos **Gráfico 1** e **Gráfico 2** permitem concluir que o algoritmo BT gasta menor tempo em relação ao BT+FC, mas realiza muito mais atribuições. A heurística MRV contribui para uma diminuição drástica da quantidade de atribuições realizadas, embora gaste mais tempo de execução realizando verificações. Em geral, a melhor alternativa é a utilização de FC+MRV, visto que atribuições desnecessárias são evitadas e a seleção da próxima célula é otimizada, buscando sempre o preenchimento da célula com menor número de valores remanescentes, ou seja, a que ocasionará menor impacto nas próximas células a serem preenchidas.