Algoritmo genético para gerência de *backlog* na metodologia SCRUM

THIAGO DOS SANTOS PINTO *

*Ciência da Computação - Graduação
E-mail: thspinto@gmail.com

***Resumo* – Projetos de engenharia de software que utilizam metodologia ágil têm *backlogs* grandes e dinâmicos. Quando times heterogêneos trabalham em um mesmo projeto há o problema de como designar uma estória aos times eficientemente. Esse trabalho criou um algoritmo genético para otimizar essa designação em termos de custo, tempo, importância e pré-requisitos das estórias do *backlog*. O algoritmo conseguiu maximizar a função de fitness adotada, mas o resultado não conseguiu atingir a meta de otimização das atribuição de estórias aos times.**

***Palavras-chave* – Método Ágil, Engenharia de Software, Planejamento**

I. INTRODUÇÃO

Até a o início da década de 1990 a metodologia de desenvolvimento de software tinha, no geral, um grande *overhead* em atividades de gerenciamento. A visão predominante era que o planejamento detalhado, utilizando-se de documentação abundante, eram essenciais para o sucesso do projeto[1].

No fim da década de 1990 iniciava-se o boom da internet, deixando o ambiente de desenvolvimento de software mais volátil por causa de requisitos dinâmicos e superficialmente especificados. Com isso, surgiram técnicas visando diminuir o *overhead* de planejamento do projeto e aumentar a resiliência do processo de engenharia de software à mudança de requisitos. Pesquisadores dessas técnicas se uniram em 2001 e publicaram o Manifesto Ágil[2], que lista os princípios de desenvolvimento dos processos denominados “ágeis”. Hoje essas técnicas são amplamente utilizadas no ambiente corporativos, sendo a metodologia SCRUM a mais popular entre elas[3].

O SCRUM é um processo iterativo que define ciclos (*sprints*) de uma a quatro semanas para o desenvolvimento, teste e entrega de *features* de um software. As *features* são representadas por “estórias”, que descrevem uma ou mais metas que o usuário deseja atingir ao utilizar o sistema. Essas estórias estão registradas no *product backlog*, que é gerenciado pelo dono do produto (*product owner*) em desenvolvimento. Segundo Schwaber e Sutherland[4], o *backlog* é um artefato vivo e em constante alteração.

Devido à dinamicidade do *backlog* e, também, ao fato de ser comum múltiplos times trabalharem em paralelo no mesmo projeto, surge um problema interessante: qual a melhor forma de distribuir as estórias do *backlog* entre os times e *sprints*? No geral isso é feito exclusivamente pela priorização feita pelo *product owner*. Será possível encontrar uma forma mais eficiente de distribuição que leve em conta a priorização do *product owner*, o tamanho da estória, a velocidade de trabalho de cada time e o custo de cada time?

Esse trabalho está organizado na seguinte forma: a seção 2 apresenta uma breve introdução a algoritmos genéticos e sua aplicação no escopo da engenharia de software. A seção 3 contém a formulação do problema e a seção 4 descreve a estratégia adotada para a solução do problema. Os resultados e a discussão são elaborados na seção 5; e as conclusões apresentadas na seção 6.

II. ALGORITMO GENÉTICO

O algoritmo genético é uma técnica de otimização que utiliza o conceito darwinista ”sobrevivência do mais apto”[5]. Nessa técnica exite uma população formada por um conjunto de soluções ao problema proposto, denominadas indivíduos. Inicialmente cria-se uma população aleatória. Então inicia-se um processo iterativo de reprodução e mutação de modo a gerar uma nova população e ampliar a variabilidade das soluções. Por fim, no final de cada iteração, consolida-se uma nova população selecionando os indivíduos mais aptos entre nova e antiga geração.

No âmbito da engenharia de software o algoritmo genético é uma ferramenta útil na otimização do gerenciamento do projeto. A atribuição eficiente de tarefas[6], por exemplo, é uma decisão complexa que envolve avaliar atributos objetivos como orçamento; e subjetivos como estimativa de tempo ou habilidade técnica. Outra forma de otimizar o projeto é avaliando as técnicas adotadas no processo de desenvolvimento[7] de forma a adapta-las às necessidades do projeto em desenvolvimento.

III. TRABALHO PROPOSTO

Este trabalho objetiva otimizar a atribuição de estórias do *backlog* ao times de desenvolvimento envolvidos no projeto. Para isso, será implementado um algoritmo genético que atribui um conjunto de estórias para cada time considerando as seguintes características:

- priorização do *product owner*;
- tamanho da estória;
- velocidade de implementação do time;
- dependência da estória (com outra estória);
- custo de desenvolvimento do time.

A. Representação dos Times

Os times T_i que trabalham no projeto terão a seguinte representação

$$T_i = \{efficiency, cost, available\_time\} \quad (1)$$

em que:

efficiency: eficiência da equipe medido em *story points* por hora trabalhada;
cost: custo da equipe medido em Reais por hora trabalhada;
available_time: tempo disponível da equipe por *sprint*.

B. Representação das Estórias

Iremos estimar o tempo de desenvolvimento a partir de *story points*, que é uma medida arbitrária do SCRUM que visa uma estimativa abstrata da complexidade da tarefa[8]. Para a prioridade da estórias iremos convencionar valores entre 0 e 1, sendo que valores maiores indicam maior prioridade.

Cada estória E_i terá a seguinte representação

$$E_i = \{deps, priority, est\_complexity, status\} \quad (2)$$

em que:

deps: vetor com estórias que são pré-requisitos para a implementação desta;
est_complexity: tempo esperado para a implementação da estória em *story points*;
priority: prioridade da estória segundo o *product owner*;
status: status da estória (no *backlog*, em desenvolvimento, entregue);

IV. Métodos

A construção do algoritmo genético foi inspirado parcialmente no trabalho de Yu e Buyya [9].

A. Indivíduos

Os indivíduos S_i do algoritmo genético são um conjunto de tuplas de time / estória, que representam a atribuição da estória ao time:

$$S_i = [(T_1, E_1), (T_2, E_2), ..., (T_i, E_i)] \quad (3)$$

B. Função de Fitness

O custo cT total dos *sprints* de todos os times é dado por

$$cT = \sum_{P_i} efficiency * est\_complexity * cost \quad (4)$$

Sendo $totalsp$ os custo total dos *sprints*, considerando todos os times, temos que o custo médio $cMed$ é

$$cMed = \frac{cT}{totalsp} \quad (5)$$

A função de fitness ft é dada por

$$ft = \frac{totalsp}{cMed} * \frac{1}{1 + (4 * invDep * excHrs)} \quad (6)$$

em que:

invDep: número de pré-requisitos da estória não presentes no mesmo *sprint*;
excHrs: número de horas excedidas do total disponível para o *sprint*

Dessa forma, a função de fitness terá maiores valores quando o $totalsp$ for maior mas será penalizada quando o custo médio aumentar, quando exceder as horas disponíveis ou quando quebrar os pré-requisitos de alguma estória.

C. Restrições

Não é permitido em um mesmo indivíduo que a mesma estória seja atribuída para diferentes times. O pré-requisito de uma estória só é considerado violado se a ele não estiver implementado e não estiver designado para o mesmo time.

D. Reprodução

Foi implementada duas estratégias de reprodução. A primeira, assexuada, apenas copia os indivíduos da população antiga.

A segunda estratégia, seleciona os dois melhores indivíduos de um groupo de C indivíduos, e faz o *crossover* entre esse indivíduos. O *crossover* escolhe uma tupla de cada indivíduo e faz a troca dessas tuplas entre os indivíduos.

Para este trabalho C foi fixado em 5.

E. Mutação

O algoritmo realiza 3 tipos de mutação, cada com com 1/3 de probabilidade de ocorrer. Esse trabalho sempre utiliza distribuição uniforme para fazer operações probabilísticas.

O primeiro tipo insere uma nova tupla estória / time na solução. A estória é escolhida aleatoriamente das estória disponíveis e o time escolhido aleatoriamente. Uma estória está disponível se ela não está implementada e não está designada a nenhum time.

O segundo tipo de mutação escolhe uma tupla qualquer do indivíduo, remove-a e cria uma nova tupla utilizando o mesmo algoritmo que o primeiro tipo de mutação.

O terceiro tipo de mutação apenas remova uma tupla do indivíduo.

F. Combinação

Duas estratégia de combinação de duas gerações de indivíduos foram implementadas.

1) **SteadyState:** os 10% melhores da nova geração de indivíduos substituem os 10% piores da antiga geração desde que indivíduo substituto seja melhor que o indivíduo substituido.
2) **Elitismo:** os 10% melhores da antiga geração de indivíduos são transferidos para a nova geração.

G. Algoritmo

O algoritmo genético segue os seguintes passos:

1) Gera a população aleatória
2) Reproduz os indivíduos
3) Cada indivíduo novo sofre mutação com uma probabilidade P
4) A antiga e a nova geração são combinadas
5) Retorna ao item 2) N vezes

Nesse trabalho foi utilizado N = 100 iterações para a execução do algoritmo.

H. Avaliação

O algoritmo foi executado no *backlog* do projeto de IoT (Apêndice A), da matéria de Engenharia de Software. Os seguintes parâmetros foram analisados:

1) Tamanho da população
2) Probabilidade de mutação
3) Tipo de reprodução
4) Tipo de combinação

A Tabela I mostra as combinações utilizadas.

Tabela I
PARÂMETROS DE EXECUÇÃO DO ALGORITMO GENÉTICO.

Tam. Pop.	Prob. Mut.	Tipo Repro.	Tipo Comb.
100	0,1	nenhuma	steadyState
100	0,8	nenhuma	steadyState
1000	0,1	nenhuma	steadyState
1000	0,8	nenhuma	steadyState
1000	0,8	nenhuma	elitismo
1000	0,1	crossover	steadyState

V. RESULTADOS E DISCUSSÃO

A. Otimização do fitness

Pelas Figuras 1 e 2, pode-se ver claramente que a baixa taxa de mutação diminui a velocidade de convergência do algoritmo.

As Figuras 1 e 2, assim como as Figuras 3 e 4 mostram o tamanho do população aumenta a velocidade de convergência. Mais indivíduos aumentam a probabilidade de se achar uma resultado melhor em menos iterações. Para populações menores pode-se aumentar o número de iterações do algoritmo, o resultado da otimização está relacionado ao número de indivíduos * número de iterações.

A taxa de mutação no caso dos experimentos das Figuras 2 e 4 não foi um fator decisivo na otimização do indivíduo com maior fitness. Os melhores indivíduos tinham fitness 170 e 173 respectivamente.

O método de combinação das gerações por elitismo apresentou resultados melhores que o por steadyState. A Figura 5 mostra a execução com combinação por elitismo. A diferença entre essa combinação e a steadyState é que esse método mantém na população indivíduos com baixo fitness. Então o valor médio entre o melhor e pior fitness não apresenta

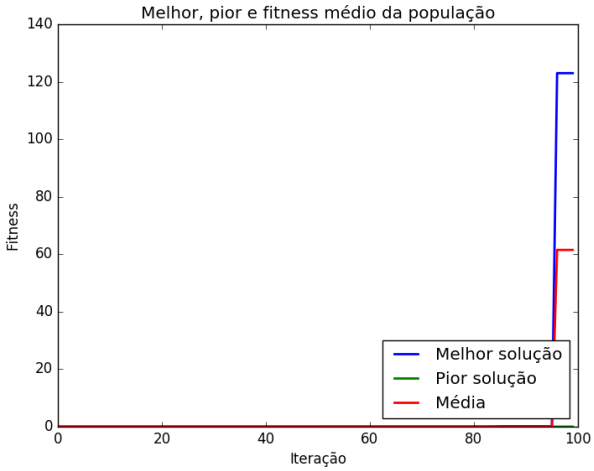

Figura 1. Resultado da execução com população 100, sem crossover, combinação steadyState e probabilidade de mutação 0,1.

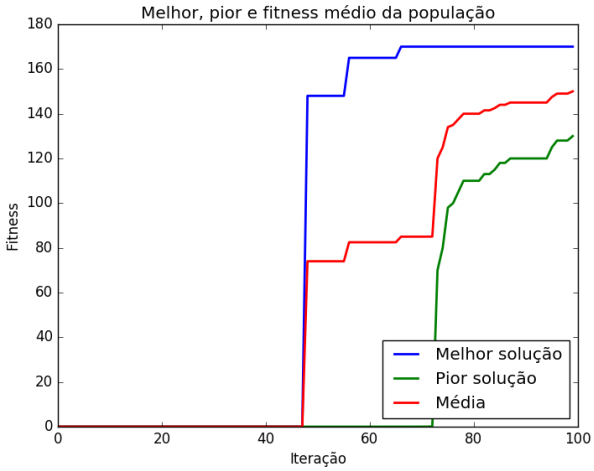

Figura 2. Resultado da execução com população 1000, sem crossover, combinação steadyState e probabilidade de mutação 0,1.

melhoria significativa. Mas o indivíduo com melhor fitness obtido no elitismo foi de 193, melhor que no caso anterior.

A Figura 6 mostra a solução com *crossover* apresenta melhor convergência. Note que o *crossover* do algoritmo altera apenas uma atribuição time → estória. Isso porque testes com alterações maiores por *crossover* resultaram na instabilidade da busca, causando um comportamento inesperado durante as iterações. Essa instabilidade provavelmente é devido às restrições do problema, grandes alterações tendem a gerar indivíduos piores. Além disso, o *crossover* tradicional é pouco eficaz nesse caso porque os indivíduos foram modelados de forma que as ordem dos ”genes”não importasse. Então ele acaba atuando como uma segunda mutação.

B. Avaliação do resultado

A melhor solução, com fitness 213, foi obtida executando o algoritmo com população 1000, com *crossover*, combinação

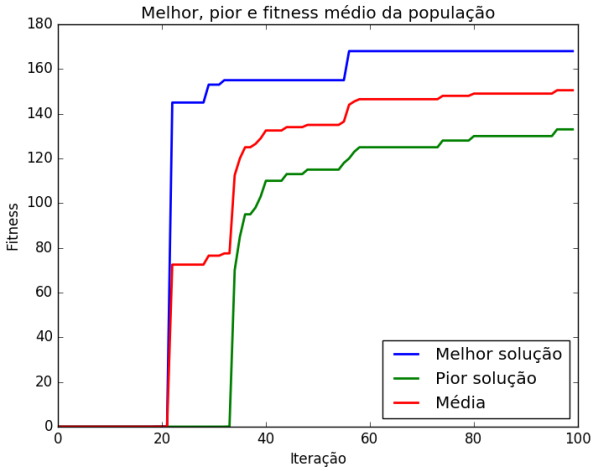

Figura 3. Resultado da execução com população 100, sem crossover, combinação steadyState e probabilidade de mutação 0,8.

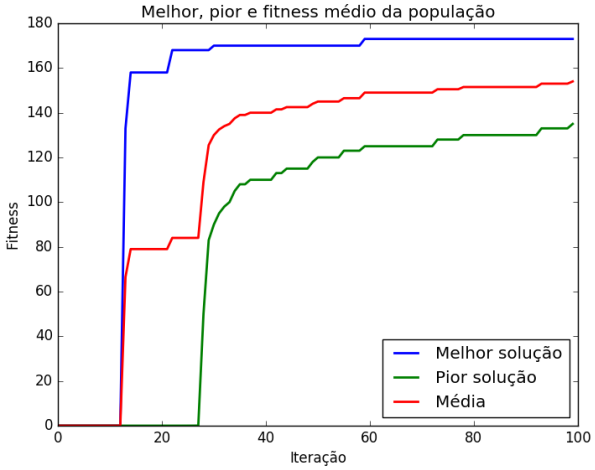

Figura 4. Resultado da execução com população 1000, sem crossover, combinação steadyState e probabilidade de mutação 0,8.

steadyState, iterações e probabilidade de mutação 0,1.

A Tabela II mostra o designação de estórias por time do melhor resultado obtido.

Tabela II
SPRINT MELHOR RESULTADO.

Time	Estórias	Total SP	Total horas
T1	S1	30	50
T2	P1, M1, R1, L1, C2	165	275
T3	J1	25	63

Comparando as Tabelas II e III pode-se observar que o algoritmo atribuiu mais tarefas ao time com mais eficiência, mas acabou subutilizando os outros times. Também, o algoritmo quebrou muitos pré-requisitos na atribuição as estórias. Isso certamente é reflexo da função da fitness que não está gerando valores ideais para a otimização que desejamos obter.

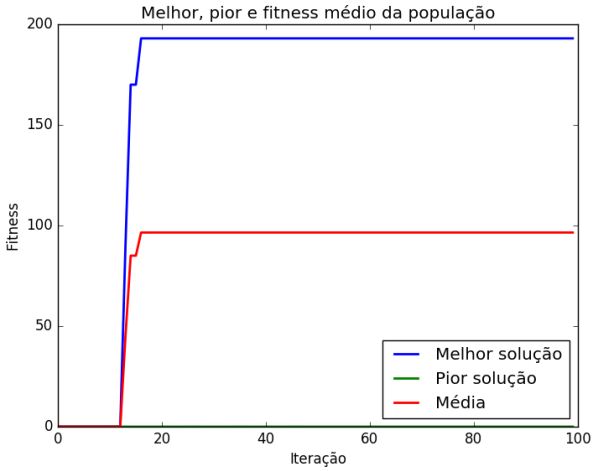

Figura 5. Resultado da execução com população 1000, sem crossover, combinação elitismo e probabilidade de mutação 0,8.

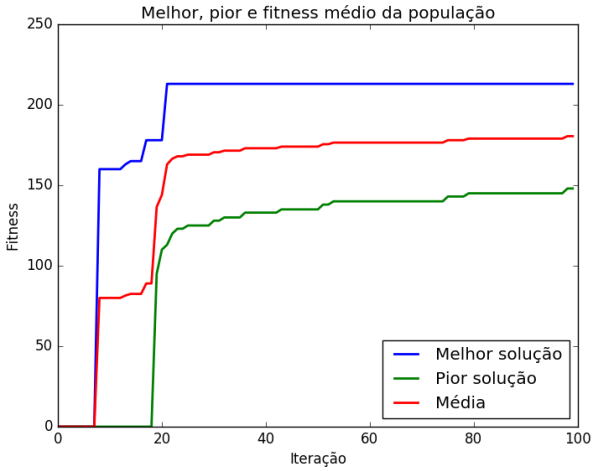

Figura 6. Resultado da execução com população 1000, com *crossover*, combinação steadyState e probabilidade de mutação 0,1.

VI. CONCLUSÕES

O trabalho mostrou uma formar abordagem diferente de se otimização o processo de desenvolvimento da metodologia SCRUM. É interessante usar algoritmos de otimização há muitos fatores que influenciam o resultado final do projeto. Nesses casos, devido à complexidade da decisão, a intuição do gerentes pode não gerar os melhores resultados.

A execução do algoritmo com o crossover simplificado foi o que resultou na melhor convergência da população. A função de fitness, porém, precisa ser reavalida pois não gerou a otimização que buscavamos nas atribuições das estórias aos times. Um time ficou sobrecarregado enquanto os outros foram subutilizados. Também, não foi evitado a quebra de pré-requisitos das estórias.

REFERÊNCIAS

[1] I. Sommerville, *Software Engineering*, ser. International Computer Science Series. Pearson, 2011. [Online]. Available: https://books.google.com.br/books?id=l0egcQAACAAJ 1

[2] J. Highsmith. (2001) History: The agile manifesto. [Online]. Available: http://www.agilemanifesto.org/history.html 1

[3] VersionOne. (2016) State of agile report. [Online]. Available: https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf 1

[4] K. Schwaber, J. Sutherland, and M. Beedle, "The definitive guide to scrum: The rules of the game," *Recuperado de: http://www. scrumguides. org/docs/scrumguide/v1/scrum-guide-us. pdf*, 2013. 1

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009, ch. 4. 1

[6] C. K. Chang, H. yi Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for software project scheduling with genetic algorithms," *Information and Software Technology*, vol. 50, no. 11, pp. 1142 – 1154, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584908000372 1

[7] R. Dwivedi and D. Gupta, "Customizing agile methods using genetic algorithm," in *Proceedings of International Conference on Advances in Communication, Network and Computing, Elsevier*, 2014. 1

[8] A. S. Greg Smith. (2009) Estimating with story points. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh273055(v=vs.88).aspx 2

[9] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3-4, pp. 217–230, 2006. 2

[10] T. dos Santos Pinto. (2016) Repositório projeto1 ia. [Online]. Available: https://github.com/thspinto/MC906A-MO416A/tree/master/project1 6

APÊNDICE A
BACKLOG E TIMES IOT (INTERNET OF THINGS)

Tabela III
TIMES DO PROJETO IOT

Id	Eficiência	Custo	Tempo Disp.
T1	0.6	50	120
T2	0.8	90	60
T3	0.4	30	160

Tabela IV
BACKLOG PROJETO IOT

Id	Descrição	Dependência	Prioridade	Tempo	Status
M1	Criar maquete para demonstrar features		0.9	8	backlog
A1	Implementar login com Oauth2		0.9	10	done
C1	Implementar Cadastro de pessoas e objetos	A1	0.9	18	backlog
C2	Implementar edição/compartilhamento de objetos	C1	0.8	10	backlog
F1	Exibir resumo do consumo das luzes da casa	A1	0.7	15	backlog
J1	Regar o jardim quando terra estiver seca e não for chover		0.6	25	backlog
P1	Abrir e fechar portas pelo celular	”C1,A1”	0.6	20	backlog
L1	Reconhecer latido do cachorro e notificar dono	A1	0.4	40	backlog
S1	Implementar sistema de sensores de movimento		0.5	30	backlog
S2	Implementar sistema de sensores de presença		0.5	35	backlog
F2	Reconhecer acesso não autorizado e avisa proprietário	”S1,A1”	0.6	40	backlog
F3	Apagar as luzes e eletrodomésticos de cômodos vazios	S2	0.8	25	backlog
I1	Mostrar todas as funcionalidades da casa e o status do componentes IoT para o usuário	”A1,C1,C2”	1	40	backlog
R1	Autenticação por voz	”A1,C1”	0.5	80	backlog
R2	Reconhecimento de comandos por voz	R1	0.6	40	backlog
L2	Ascender lâmpada por presença e luz	S2	0.3	10	backlog
D2	Desgarga inteligente (economizar água)		0.2	15	working
M1	Monitorar consumo de água	C1	0.6	15	backlog
U1	Diminuir aquecimento/ar condicionado quando não tiver pessoas na casa	S2	0.6	20	backlog
U2	Prever quando moradores forem chegar na casa e ajustar temperatura da casa	U1	0.5	50	backlog

Dados estão disponíveis em CSV junto com o código fonte em [10].