# Aplicando o Algoritmo *Ant-Q* na Priorização de Requisitos de Software com Precedência

**Matheus Henrique Esteves Paixão[1], Márcia Maria Albuquerque Brasil [1], Thiago Gomes Nepomuceno da Silva [1], Jerffeson Teixeira de Souza[1]**

[1]Grupo de Otimização em Engenharia de Software da UECE –
Universidade Estadual do Ceará (UECE)
Avenida Paranjana, 1700 – Fortaleza – CE - Brasil

{mhepaixao,marcia.abrasil,thi.nepo}@gmail.com, jeff@larces.uece.br

***Abstract.*** *Software Requirements Prioritization is related to deciding the sequence for requirements development. It should be done during the project planning. For this task, some aspects should be taken into consideration, especially meet customers' expectations, address the risk for the project and comply with requirements interdependencies. Ant colony based approaches have been prominent as a search strategy for solving Software Engineering problems. In this work, the Ant-Q algorithm is adapted for solving the software requirements prioritization problem with precedence. Experiments are carried out for solving problem instances.*

***Resumo.*** *A priorização de requisitos de software envolve decidir em que seqüência as funcionalidades devem ser desenvolvidas e deve ser realizada durante a fase de planejamento do projeto. Para tanto, alguns aspectos devem ser considerados, destacando-se o atendimento às expectativas dos clientes, os riscos para o projeto e as interdependências entre os requisitos. Algoritmos baseados em colônia de formigas vêm se destacando como mais uma estratégia para resolução de problemas da Engenharia de Software por meio de busca. Neste trabalho, uma adaptação do algoritmo Ant-Q é realizada para resolver o problema da priorização de requisitos com precedência. Experimentos são realizados na resolução de instâncias do problema.*

## 1. Introdução

Uma vez que os produtos de software desempenham um papel fundamental nos processos de negócio, é de extrema importância que funcionem de acordo com as necessidades e objetivos para os quais foram desenvolvidos. Assim, a correta identificação e o entendimento dos principais requisitos de negócio representam um importante desafio para empresas de software a fim de assegurar que os produtos desenvolvidos atendam às expectativas de seus clientes.

No entanto, dado um conjunto de funcionalidades a serem desenvolvidas, planejar e definir em que ordem os requisitos devem ser implementados não é uma tarefa simples. Essa dificuldade advém do fato de que é preciso considerar as prioridades de cada cliente, o risco associado a cada requisito (que pode impactar no projeto como um todo) e o fato de que os requisitos podem estar relacionados entre si de alguma forma. Esse problema pode ser denominado como problema da priorização de requisitos de software. Neste caso, a definição de uma seqüência de implementação é necessária, pois permite um melhor gerenciamento dos riscos do projeto, assegurando

que os requisitos com um maior risco associado são tratados logo no início. Ou ainda, garante que, caso o projeto precise ser interrompido por um motivo qualquer (falta de recursos, por exemplo), os requisitos mais importantes já foram implementados.

Algoritmos de otimização vêm sendo aplicados na solução de problemas da Engenharia de Software caracterizados, muitas vezes, pela existência de restrições, objetivos conflitantes e grandes espaços de busca. Denominada *Search-based Software Engineering – SBSE* [Harman e Jones 2001], essa área de pesquisa sugere que muitos problemas da Engenharia de Software, devido a sua natureza complexa, podem ser reformulados como um problema de busca tornando-se, assim, adequados para a aplicação de técnicas de otimização, notadamente as metaheurísticas. Dentre as metaheurísticas que vêm sendo aplicadas com sucesso em SBSE, pode-se destacar a Otimização por Colônia de Formigas (*ACO*) [Dorigo et al. 1996]. O algoritmo é inspirado no comportamento natural das formigas em sua busca por fontes de alimento. Durante essa busca, o processo de comunicação entre as formigas ocorre de forma indireta por meio de trilhas de feromônio.

Assim, a principal contribuição desta pesquisa consiste em apresentar uma adaptação do algoritmo *Ant-Q* [Gambardella e Dorigo 1995] a ser aplicado ao problema da priorização de requisitos de software com precedência. O *Ant-Q* é um algoritmo de colônia de formigas que utiliza uma técnica de aprendizado por reforço denominada *Q-Learning*. Além desta seção introdutória, o trabalho é organizado conforme a seguir. Seção 2, descreve alguns trabalhos relacionados ao problema aqui abordado; Seção 3, define formalmente a abordagem proposta e descreve importantes aspectos levados em consideração; Seção 4, apresenta o algoritmo *Ant-Q*; Seção 5, mostra como o algoritmo foi adaptado para resolver o problema da priorização de requisitos; Seção 6, relata experimentos realizados, onde o desempenho do *Ant-Q* é comparado com o de outros algoritmos; e, finalmente, Seção 7, apresenta as considerações finais do trabalho.

## 2. Trabalhos Relacionados

A seleção de requisitos é formulada como um problema de otimização em [Bagnallet al., 2001], sendo o trabalho conhecido na área de SBSE como o Problema do Próximo *Release* (*The Next Release Problem*– NRP). A abordagem consiste em selecionar, através de uma formulação mono-objetiva, um conjunto ótimo de clientes, cujos requisitos deverão ser disponibilizados na próxima versão do sistema. Os recursos disponíveis e a precedência entre os requisitos são restrições para o problema. Uma abordagem multiobjetiva do problema é apresentada por Zhang et al [2007], onde a satisfação de clientes e o custo total do projeto são os objetivos a serem otimizados. A formulação, contudo, não inclui qualquer forma de interdependência entre requisitos, algo incomum no contexto de projetos reais.

O planejamento de *releases*, definição de estágios de entrega de funcionalidades do sistema, é abordado nos trabalhos de Greer e Ruhe [2004], Ruhe e Saliu [2005] e Colares et al. [2009]; onde valor, risco e prioridade do requisito, interdependência, importância do cliente e recursos disponíveis são alguns dos fatores considerados.

O conjunto das técnicas utilizadas para resolver o problema do próximo *release* é ampliado e uma adaptação do algoritmo *Ant Colony System* – ACS é aplicada em [Del Sagrado et al. 2010]. Metaheurísticas como Algoritmos Genéticos e Têmpera Simulada também foram utilizadas. Interdependências entre requisitos não foram consideradas.

A metaheurística *Ant Colony System* também foi adaptada para resolver o problema do planejamento de *releases* com requisitos interdependentes em [Souza et al. 2011]. Os recursos disponíveis em cada *release* são limitações do problema e os resultados obtidos são comparados com Algoritmos Genéticos e Têmpera Simulada.

## 3. Problema da Priorização de Requisitos de Software

Esta seção apresenta uma descrição formal do problema e define alguns aspectos relacionados à abordagem proposta que foram considerados na modelagem e na implementação do algoritmo.

Requisitos consistem em descrições de como um sistema de software deve funcionar. São definidos nos estágios iniciais do desenvolvimento e consistem em uma especificação do que deve ser implementado sendo, portanto, um conjunto de necessidades que devem ser atendidas.

Assim, seja $R = \{r_i | \, i = 1, 2, \ldots, N\}$ um conjunto de requisitos a serem desenvolvidos. Cada requisito $r_i$ possui um risco associado, representado por $risk_i$, que pode ser definido em termos da análise de impacto do risco para o negócio do cliente *versus* a sua probabilidade de ocorrência. Desta forma, o risco de cada requisito varia em uma escala de 1 (mais baixo risco) a 9 (mais alto risco), sendo quantificado pela Tabela 1 a seguir. Ainda, os requisitos podem estar relacionados entre si de diversas formas. O tipo de relacionamento aqui considerado é a precedência técnica entre requisitos, identificada quando a implementação de certo requisito pressupõe o prévio desenvolvimento de um ou mais requisitos.

**Tabela 1. Quantificação do Risco: Impacto *versus* Probabilidade de Ocorrência**

| Impacto no Negócio | Probabilidade | | |
|---|---|---|---|
| | Baixo | Médio | Alto |
| Baixo | (1) | (2) | (3) |
| Médio | (4) | (5) | (6) |
| Alto | (7) | (8) | (9) |

*Stakeholders* exercem um importante papel na priorização de requisitos. Seja $C = \{c_m | \, m = 1, 2, \ldots, M\}$ o conjunto de *stakeholders* envolvidos com o desenvolvimento do sistema. Para cada *stakeholder* $c_m$, é associado um peso, representado por $w_m$, baseado em sua importância relativa para a organização e sendo quantificado em uma escala de 1 (baixa importância) a 10 (alta importância). Neste trabalho, o conjunto considerado envolve apenas clientes. Diferentes clientes possuem diferentes interesses com a implementação de cada requisito. O conceito de valor do requisito é aqui analisado do ponto de vista de cada cliente e de sua importância para a organização. Assim, $value(m, i)$ quantifica a importância que um *stakeholder* $c_m$ associa a um requisito $r_i$, atribuindo um valor que varia de 1 (baixa importância) a 10 (alta importância).

A finalidade da priorização aqui proposta é ordenar os requisitos de acordo com os objetivos de valor (maximização do valor agregado ao negócio da organização, baseado no valor dos requisitos e dos clientes mais importantes) e risco (minimização dos riscos do projeto como um todo), observando a precedência técnica existente entre os requisitos. Assim, deve-se estabelecer uma ordem de implementação, onde os

requisitos considerados mais importantes, do ponto de vista dos clientes mais significativos, e os que apresentarem maior risco devem ser desenvolvidos primeiro.

Assim, o problema é formulado matematicamente conforme a seguir:

$$\boldsymbol{max} \sum_{\boldsymbol{i=0}}^{\boldsymbol{N-1}} (\boldsymbol{N-i}) * (\boldsymbol{\rho} * \boldsymbol{risk_i} + \boldsymbol{\sigma} * \boldsymbol{score_i})$$

$$\text{onde, } score_i = \sum_{j=1}^{M} value(j,i) * w_j$$

Na formulação acima as constantes $\boldsymbol{\rho}$ e $\boldsymbol{\sigma}$ representam os pesos atribuídos ao risco e à importância respectivamente. Vale ressaltar ainda que, quanto antes um requisito for priorizado, maior será a sua contribuição para a função objetivo.

## 4. Algoritmo *Ant-Q*

O primeiro algoritmo da Otimização por Colônia de Formigas (ACO) foi proposto por Dorigo e Maniezzo (1996). Denominado *Ant System*, era inspirado no comportamento natural das formigas na busca por boas fontes de alimento. A comunicação entre formigas é indireta, feita a partir de trilhas de feromônio. Quando uma formiga encontra uma fonte de comida, deixa uma trilha de feromônio para servir de guia para as demais formigas. Quanto melhor for a trilha, maior será a quantidade de feromônio e melhor será a fonte de alimento. A partir do *Ant System* e da técnica de aprendizado por reforço *Q-Learning*, foi desenvolvido o *Ant-Q* [Gambardella e Dorigo 1995]. Neste algoritmo uma formiga situada no nó $r$ move-se para o nó $u$ de acordo com a seguinte regra de transição:

$$(1)\quad s = \begin{cases} \arg max_{u \in J_k(r)} \{[AQ(r,u)]^{\delta}.[HE(r,u)]^{\beta}\} & se\ q \leq q_0 \\ S & senão \end{cases}$$

O valor $AQ(r,u)$ representa a quantidade de feromônio na aresta $(r,u)$ e o valor $HE(r,u)$ é o seu valor heurístico. Esses valores representam a qualidade da aresta segundo o que já foi aprendido pelas formigas e segundo a natureza do problema, respectivamente. Os valores $\delta$ e $\beta$correspondem aos seus devidos pesos. No momento da transição, é gerado um número aleatório no intervalo $(0,1)$ que, quando comparado com a constante $q_0$, decidirá se a formiga $k$ irá para o melhor nó possível dentre todos os que ainda não foram visitados, $J_k(r)$, ou se tomará uma decisão pseudo-aleatória $S$. Caso a formiga $k$ se mova de forma pseudo-aleatória, a probabilidade de ir para o nó $s$ é dada pela seguinte expressão:

$$(2) p_k(r,s) = \begin{cases} \dfrac{[AQ(r,s)]^{\delta}.[HE(r,s)]^{\beta}}{\sum_{z \in J_k(r)}[AQ(r,z)]^{\delta}.[HE(r,z)]^{\beta}} & se\ s \in J_k(r) \\ 0 & senão \end{cases}$$

O objetivo é fazer com que os melhores nós tenham uma maior probabilidade de transição. Após todas as formigas terminarem suas respectivas rotas por todos os nós, calcula-se o valor de reforço baseado na qualidade da solução encontrada. Esse valor é calculado a partir da melhor solução dentre todas as formigas e é aplicado somente às arestas pertencentes a esta melhor solução. O valor de reforço $\Delta AQ(r,s)$ é dado por:

$$(3)\quad \Delta AQ(r,s) = w * L_{kgb}$$

O valor da solução encontrada pela melhor formiga é dado por $L_{kgb}$ e $w$ é um fator de normalização. Detalhes sobre a utilização do valor de reforço e a atualização do feromônio são discutidos na próxima seção.

## 5. Adaptação do algoritmo *Ant-Q* para resolução do problema

O *Ant-Q* será utilizado para resolução do problema de priorização de requisitos de software levando em consideração a precedência técnica entre os requisitos. O objetivo é maximizar o valor e minimizar os riscos do projeto. Para isso, os requisitos mais importantes e com maior risco devem ser implementados primeiro. Desta forma, o valor heurístico da transição do requisito $r$ para o requisito $u$ é dado por:

$$(4)\quad HE\ (r,u) = \frac{score_u + risk_u}{MaxHE}$$

$MaxHE$ representa o maior valor heurístico possível e serve para efeito de normalização. Inspirado por [Souza et al 2011], ao fazer a transição do requisito $r$ para o requisito $u$, a formiga atualiza o feromônio, ou valor *Ant-Q*, dessa aresta da seguinte forma:

$$(5)\quad AQ(r,u) = (1-\alpha) * AQ(r,u) + \alpha * (\Delta AQ(r,u) + \gamma * AQ_0)$$

Onde $\alpha$ é o fator de evaporação do feromônio, $\gamma$ é o desconto de aprendizagem e $AQ_0$ é o feromônio inicial. A seguir o seu pseudocódigo e a descrição do algoritmo.

**Algoritmo 1: Adaptação do AntQ para o Problema da Priorização de Requisitos**

```
Inicialização
Inicializar constantes
for each aresta do
   Atribuir feromônio inicial AQ0
end for
for each requisito sem precedente do
   Colocar uma formiga
end for

Laço principal
while Condição de parada do
   for each requisito do
      Escolher novo requisito a partir de (1) e (2)
      if requisito tem precedentes then
         adicionarPrecendentes()
      end if
      Atualizar feromônio de acordo com (5) (ΔAQ(r,s) = 0)
   end for
   Obter melhor solução Lkgb dentre todas as formigas
   Calcular ΔAQ a partir de (3)
   Atualizar feromônio de todas as arestas Є Lkgb de acordo com (5) (AQ0 = 0 )
end while
return Melhor solução

adicionarPrecedentes()
for each precedente do
   if precedente tem precedentes then
      adicionarPrecedentes()
   end if
end for
```

Na fase inicial, são configuradas as constantes de inicialização, uma formiga é colocada em cada requisito que não apresenta nenhum precedente e é atribuído a cada aresta um feromônio inicial $AQ_0$. No laço principal, cada formiga se move para um novo requisito de acordo com (1) e (2). Se o requisito escolhido tiver algum precedente que ainda não foi visitado, adicionam-se todos os precedentes deste requisito, bem como os precedentes dos precedentes de forma recursiva. Somente após todos os precedentes adicionados, a formiga se move para o requisito escolhido inicialmente. Após todas as formigas terem feito a escolha do próximo requisito, as mesmas atualizam o feromônio da aresta percorrida de acordo com (5). Quando as formigas terminarem de percorrer todos os requisitos, seleciona-se a melhor solução dentre todas as formigas, calcula-se o valor de reforço $\Delta AQ$ utilizando (3) e atualiza-se o feromônio das arestas pertencentes à melhor solução, novamente de acordo com (5). Após atingir um critério de parada, é retornada a melhor solução encontrada ao longo das iterações.

## 6. Avaliação Empírica

Uma avaliação empírica foi conduzida com o objetivo de demonstrar a aplicação e a eficiência do algoritmo *Ant-Q*, conforme adaptação apresentada, na resolução de instâncias do problema modelado. O desempenho do algoritmo é comparado com o de um algoritmo genético e de um algoritmo randômico. Esta seção descreve o contexto utilizado nos experimentos, bem como apresenta e analisa os resultados obtidos.

### 6.1. Descrição das Instâncias

Nove instâncias de problemas foram geradas e utilizadas para avaliar a modelagem do problema e a aplicação dos algoritmos em contextos distintos. São elas: InstA (50 requisitos, 3 clientes), InstB (50,5), InstC (50,8), InstD (100,3), InstE (100,5), InstF (100,8), InstG (200,3), InstH (200,5), InstI (200,8). Todas as instâncias apresentam nível de precedência de 10% e parâmetros $\rho$ e $\sigma$ iguais a 1. Os valores para $risk_i, w_m$, e $value(m,i)$ foram gerados aleatoriamente de acordo com as escalas definidas anteriormente. Matrizes de Precedência Técnica também foram geradas randomicamente obedecendo aos percentuais estabelecidos.

### 6.2. Algoritmo Genético e Algoritmo de Busca Aleatória

Os algoritmos genéticos são algoritmos evolucionários extensamente utilizados, são inspirados na teoria da evolução das espécies de Darwin, que simulam processos biológicos como mutações, cruzamentos e seleção. [Goldberg, 1989]

Neste trabalho, um algoritmo de busca aleatória também foi utilizado como um referencial para permitir que os resultados obtidos pelas metaheurísticas pudessem ser validados. Qualquer metaheurística deve ser capaz de superar um algoritmo de busca aleatória para ser aplicada com sucesso em SBSE [Harman e Jones, 2001].

### 6.3. Configuração dos Parâmetros dos Algoritmos

Os parâmetros utilizados para cada metaheurística foram configurados a partir da execução de testes preliminares. Assim, para o Algoritmo Genético foram utilizados: tamanho da população igual a 100 indivíduos (a população inicial foi gerada aleatoriamente), taxa de cruzamento igual a 0,9 utilizando o operador *TwoPointsCrossover*, taxa de mutação igual a (1 ÷ número de requisitos) utilizando o

operador *SwapMutation* e seleção utilizando o método torneio binário. Para o Ant-Q foram utilizados: $\delta$ igual a 1, $\beta$ igual a 2, $\alpha$ igual a 0.1, $\gamma$ igual a 1 e $q0$ igual a 0.9. Todos os algoritmos foram configurados para realizarem um total de 500 avaliações.

### 6.4. Apresentação e Análise dos Resultados

Neste trabalho, foram calculados a média e o desvio-padrão de dez execuções de cada algoritmo em cada instância para definir o comportamento da função-objetivo do problema e do tempo de execução (em milissegundos). Os resultados obtidos em cada instância são apresentados na tabela a seguir.

**Tabela 2. Resultados**

| | Valor da função objetivo | | | Tempo | | |
|---|---|---|---|---|---|---|
| Instância | *Ant-Q* | GA | Aleatório | *Ant-Q* | GA | Aleatório |
| InstA | 87094±23,70 | 81472,3±355,56 | 79902,8 ± 612,29 | 1962,3±11,06 | 3190,7 ± 195,96 | 17017,2 ± 1001,62 |
| InstB | 277400,7±84,76 | 263611 ± 2497,06 | 259211,4 ± 2166,57 | 1973,8±19,22 | 1090,7 ± 139,45 | 6136,1±385,5 |
| InstC | 385605,9±14,59 | 371371,9 ± 1755,92 | 367331,4 ± 724,11 | 2181,8±10,74 | 1009,5 ± 98,22 | 5557,7 ± 289,39 |
| InstD | 573533,3±75,41 | 530711,8±3791,24 | 522590,5 ± 2557,88 | 15716,5±111,76 | 41291,6 ± 3160,97 | 219511,4 ± 12982,76 |
| InstE | 1110099,7±78,19 | 1029170,3 ± 4558,99 | 1003608,6 ± 2677,15 | 15640,7±50,80 | 63653,1 ± 7777,41 | 107308,5 ± 10677,15 |
| InstF | 1778108,8±81,76 | 1704041,1±6149,46 | 1687979,6±2385,31 | 16294,8±210,56 | 49468,9 ± 6059,72 | 252059,1±11824,3 |
| InstG | 3423202,1±275,16 | 3051235,3 ± 18087,74 | 3006202,7±25487,2 | 145026,6±787,33 | 227389,1 ±24819,51 | 1173279,7 ±31248,8 |
| InstH | 2471667±71,36 | 2257666,9 ± 10421,18 | 2238250 ± 22182,5 | 143307,2±1624,21 | 238294,7 ± 32383,27 | 1481708 ± 32654,9 |
| InstI | 6394383,3±521,49 | 6009152 ± 24204,18 | 5987107 ± 29417,6 | 149320,8±757,35 | 269947,9 ± 22744,4 | 1542111,4±38716,6 |

A partir da tabela apresentada percebe-se que o *Ant-Q* demonstra melhor desempenho, na maioria das instâncias, tanto para os valores da função objetivo como no tempo de execução. Com relação ao Algoritmo Genético, o *Ant-Q* apresenta resultados, em média, 6.57% e 20% melhores em comparação com a função objetivo e o tempo de execução, respectivamente. Comparando com a busca aleatória esses valores chegam a 7% e 74%. Devido às restrições de precedência entre os requisitos, o Algoritmo Genético demora muito tempo para encontrar sua população inicial, o que influencia muito no seu tempo de execução, restrição esta que não afeta o algoritmo proposto, pois o mesmo não permite que uma formiga chegue a uma solução inválida. Vale ressaltar ainda o baixo desvio-padrão do *Ant-Q*, chegando a ser 98% menor que o do Algoritmo Genético e 99% menor que o Algoritmo Aleatório, em média.

## 7. Considerações Finais

O algoritmo proposto apresentou resultados melhores que o Algoritmo Genético tanto para o valor da função objetivo como para o tempo de execução na grande maioria das instâncias testadas. Devido a esses resultados pretende-se abordar novos relacionamentos entre os requisitos e não somente precedência técnica, bem como utilizar este algoritmo para tentar solucionar outros problemas da *SBSE*. Como trabalho futuro, também pretende-se desenvolver um algoritmo baseado no Ant-Q para solução de problemas multiobjetivos.

## Referências


Bagnall, A. J., Rayward-Smith, V. J. and Whittley, I. M. (2001), "The Next Release Problem", *Information and Software Technology*, 43(14), 883–890.

Colares, F., Souza, J., Carmo R., Pádua, C. e Mateus, G. R. (2009), "A New Approach to the Software Release Planning", *Proceedings of the XXIII Brazilian Symposium on Software Engineering*– SBES '09, 207-215.

del Sagrado, J., del Águilla, I. M. and Orellana, F. J. (2010), "Ant Colony Optimization for the Next Release Problem: A Comparative Study", *Proceedings of the 2nd International Symposium on Search Based Software Engineering – SSBSE '10*, 67-76, IEEE Computer Society.

Dorigo, M., Maniezzo, V., Colorni, A. (1996), "The Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man and Cybernetics-Part B*, 26(1), 29-41.

Gambardella, L. M., Dorigo, M. (1995), "Ant-Q: A Reinforcement Learning approach to the traveling salesman problem", *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, 252-260, Tahoe City, CA. Morgan Kaufmann.

Goldberg, D. E., "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley*, 1989.

Greer D. e Ruhe, G. (2004), "Software Release Planning: An Evolutionary and Iterative Approach", *Information and Software Technology*, 46(4), 243-253.

Harman, M. and Jones, B. F. (2001), "Search-Based Software Engineering", *Information & Software Technology*, 43(14), 833-839.

Ruhe, G. e Saliu, O. (2005), "The Art and Science of Software Release Planning", *IEEE Software*, 22(6), 47-53.

Souza, J., Maia, C., Ferreira, T., Carmo, F., Brasil, M. (2011), "An Ant Colony Optimization Approach to the Software Release Planning with Dependent Requirements", *Proceedings of the 3nd International Symposium on Search Based Software Engineering (SSBSE '11)*, 142-157, Szeged, Hungary. Springer-Verlag.

Zhang Y., Harman, M. and Mansouri, S. A. (2007), "The Multi-Objective Next Release Problem", *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, 1129–1137, London, UK.ACM.