Simplificando o Balanceamento de Atributos em RPGs Eletrônicos

Alexandre Magno Monteiro Santos
Instituto UFC Virtual
Universidade Federal do Ceará
Fortaleza, Brasil
magnomont12@gmail.com

Paulo Bruno de Sousa Serafim
Departamento de Computação
Universidade Federal do Ceará
Fortaleza, Brasil
paulobruno@alu.ufc.br

Artur de Oliveira da Rocha Franco
Departamento de Computação
Universidade Federal do Ceará
Fortaleza, Brasil
artur.fhtagn@gmail.com

Rafael Augusto Ferreira do Carmo
Instituto UFC Virtual
Universidade Federal do Ceará
Fortaleza, Brasil
carmorafael@virtual.ufc.br

José Gilvan Rodrigues Maia
Instituto UFC Virtual
Universidade Federal do Ceará
Fortaleza, Brasil
gilvanmaia@virtual.ufc.br

***Abstract*—A criação de jogos do gênero RPG requer diversas etapas que estão relacionadas com os seus sistemas. No sistema de combate, a definição das habilidades dos personagens é uma tarefa importante e que muitas vezes envolve balanceamento manual por parte do game designer. Neste trabalho, é proposto um processo de balanceamento automático de atributos de personagens em jogos de RPG para simplificar o processo de criação de personagens e otimizar o tempo utilizado no balanceamento. Para alcançar tais resultados, utiliza-se algorítimo genético para identificação de parâmetros da curva de crescimento de um jogador para que este alcance uma taxa de vitórias pré-determinada contra um inimigo previamente criado pelo game designer. A ferramenta de balanceamento proposta foi capaz de gerar os atributos do personagem para cada um dos níveis dentro da margem de erro definida. Em seguida, foram geradas as curvas de nível iniciais, que são suavizadas para gerar as curvas finais. Uma avaliação experimental utilizou dez níveis de um inimigo com uma taxa de vitória desejada de 80% e margem de erro de 5%. Esses resultados sugerem que o uso do algoritmo genético foi eficaz na geração de curvas de nível, sendo adequada como um processo de balanceamento automático para auxiliar o game designer.**

***Keywords*-balanceamento automático, sistema de combate de RPG, algoritmo genético**

I. Introdução

Um dos elementos centrais dos jogos de *Role Playing Game* (RPG) estão as suas regras, os chamados sistemas [1], os quais definem parte do *gameplay* e das interações com o cenário. Esses sistemas costumam oferecer *progressões* de níveis e um certo grau de customização dos personagens aos jogadores. Uma dessas progressões é a chamada *curva de nível*, que apresenta os valores das habilidades de uma entidade para cada nível. Definir essas progressões prova-se muitas vezes como um desafio ao game designer quando do planejamento das representações adequadas dos personagens dos jogadores e dos inimigos para cada nível [2]. Uma modelagem inadequada pode representar uma frustração na experiência do jogador, seja pelo tédio de ser muito fácil, seja pela ansiedade por um desafio muito difícil [3].

Um dos elementos cobertos comumente em um sistema é o combate. De fato, boa parte do desafio dos jogos envolve o confronto do jogador com personagens controlados pela máquina ou por outro participante. Em jogos baseados em turno, essa experiência torna-se menos difícil de mensurar, uma vez que as representações de habilidade permitem simulações de batalhas e assim obter uma estimativa da taxa de vitórias e derrotas de cada um dos oponentes. Mais ainda, é possível determinar a melhor distribuição de recursos para a progressão dentro do jogo. Dessa forma, para manter um combate balanceado é de grande importância que o game designer construa uma progressão adequada das habilidades tanto dos jogadores quanto dos inimigos.

Dada a sua importância, a definição dos atributos dos jogadores é um processo realizado com bastante cuidado. Comumente esse processo é realizado manualmente e por vezes com base na tentativa e erro [3]. Dessa forma, estratégias de teste e validação das representações de personagens são recursos importantes ao processo de game design. No entanto, elas podem ser bastante custosas, apesar de indicar se há problemas que gerem dúvidas, ambiguidades, estratégias dominantes, entre outras [3].

Trabalhos envolvendo jogabilidade e jogos digitais costumam ter um apelo forte na área da Educação, Saúde, Publicidade e Cultura. Estudos que buscam compreender ou melhorar os processos de design são relativamente novos e estão mais vinculados a trabalhos de designers experientes e consagrados em discussões publicadas como livros [3], [4] do que em revisões sistemáticas pela comunidade acadêmica [5]. Ainda assim, há trabalhos pioneiros recentes que exploram meandros das mecânicas, como Maranhão et al. [5], que buscam avaliações estéticas envolvendo elementos específicos dos personagens, como Islam et al. [6], ou ainda em Faria e Pereira [7], onde são gerados personagens para *Tabletop Role*

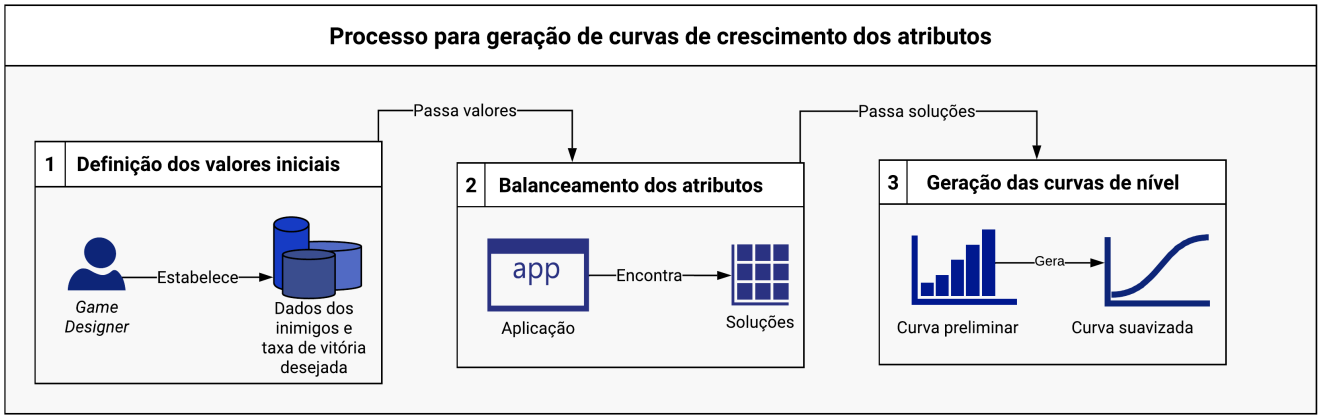

Fig. 1. Visualização do processo de balanceamento automático e geração de curvas de nível dos atributos. (1) A definição dos dados dos inimigos e as taxas de vitória desejadas são estabelecidas previamente pelo game designer. (2) A aplicação executa o balanceamento e retorna os valores de cada nível. (3) A partir dos valores encontrados automaticamente pelo algoritmo genético, é criada uma curva de nível preliminar, que é suavizada em seguida. A curva suavizada é o resultado final do processo e pode ser usada em outras ferramentas externas, como o RPG Maker MV. Fonte: os autores

Playing Game (TRPG) agradando a percepção de jogadores no contexto do jogo.

Alguns trabalhos buscam explorar essa problemática envolvendo representações de balanceamentos em contextos de progressões de equipamentos [8] e balanceamento através de algoritmos evolucionários [9]. A ideia de automatização de testes também é abordado por [9], onde a Inteligência Artificial (IA) simula combates *Player versus Player* (PvP) ou *Player versus Environment* (PvE). Outros trabalhos envolvendo modelos de jogos e suas representações já fazem buscas com outros modelos, sejam eles para narrativas [10] ou para encontrar soluções para *puzzles* [11]. Contudo, o balanceamento do jogo não trata apenas de simulações para empates, mas de um processo integrado que consiga conduzir as experiências do jogador, o que inclui o controle da dificuldade, expectativa e aprendizado, dentre outros.

Neste trabalho é apresentada uma técnica de geração de valores de habilidades de um jogador a partir dos atributos de um inimigo hipotético e a taxa de vitória desejada em um combate direto entre eles. Dessa forma, visa-se eliminar do game designer a tarefa de definir manualmente os atributos de um jogador, evitando um processo laborioso e suscetível a várias iterações de um custoso processo de tentativa e erro. Para tanto, será utilizada uma estratégia de otimização via Algoritmo Genético (AG) [12] de modo a encontrar soluções em espaços de buscas através de simulações de batalhas.

Foi desenvolvido um caso de estudo com a técnica de balanceamento automático utilizando RPG Maker MV[1] para validar a técnica proposta. As definições de habilidades e o modelo de combate já prontos na ferramenta foram utilizados para ilustrar um exemplo gerado pelo algoritmo genético. Por fim, são mostrados os resultados na interface do programa e as simulações realizadas. A representação geral do processo de balanceamento automático dos atributos e geração de curvas de níveis é mostrada na Fig. 1.

Este artigo está organizado da seguinte forma. A Seção II explica brevemente a técnica de algoritmo genético, essencial para o entendimento desta proposta. Em seguida, a Seção III apresenta trabalhos relacionados encontrados durante a revisão de literatura deste projeto. Na Seção IV, são apresentadas as regras do modelo utilizado, bem como os experimentos realizados. A Seção V apresenta os resultados obtidos após a execução da técnica descrita e a discussão a seu respeito. Por fim, a Seção VI conclui o artigo e apresenta perspectivas futuras das oportunidades de pesquisa a partir deste trabalho.

II. Referencial Teórico

Esta seção traz uma breve explicação do conceito de Algoritmo Genético, técnica de otimização utilizada para gerar automaticamente os valores de habilidades de um jogador com base nos atributos de um oponente e da taxa de vitória esperada. A partir dos resultados gerados pelo algoritmo genético, nossa abordagem constrói as curvas de níveis de cada atributo do jogador.

Os AGs foram criados por John Holland e são inspirados nos princípios das ideias de Charles Darwin sobre seleção natural [12]. Eles fornecem recursos probabilísticos e otimizações paralelas, que têm seu principio nas leis de seleção natural e sobrevivência do indivíduo mais forte [13]. São algoritmos robustos, genéricos e facilmente adaptáveis, sendo amplamente estudados e com diversas aplicações em diferentes áreas, pois podem ser utilizados desde que se conheça uma forma objetiva de comparar duas possíveis soluções para um problema, ainda que se desconheça um processo claro para a geração dessas soluções.

Mitchell [14] afirma que não há uma definição rigorosa de algoritmo genético aceito por toda a comunidade. No entanto, a maioria dos métodos possuem os seguintes elementos em comum:

- População de cromossomos;

[1]https://www.rpgmakerweb.com/products/programs/rpg-maker-mv

- Seleção através de uma função objetivo;
- Cruzamento para a geração de novos indivíduos; e
- Mutações aleatórias dos novos indivíduos.

O termo cromossomo refere-se tipicamente a uma solução candidata para o problema, representada por uma *string* de bits e que pode ser visto como um ponto no espaço de busca de soluções. A função objetivo atribui um escore para cada cromossomo, que depende de quão apropriada é a solução deste indivíduo, daí o uso do termo *fitness function*, inspirado pelo conceito Darwiniano. O cruzamento consiste na troca de material genético entre dois indivíduos, computacionalmente representada pela geração de novos indivíduos a partir da combinação de dois já existentes. Por fim, a operação de mutação modifica aleatoriamente alguns dos bits dos cromossomos [14], o que confere maior diversidade à busca.

Um algoritmo genético simples segue as etapas descritas a seguir [14]:

1) Iniciar com uma população de N cromossomos gerada aleatoriamente, onde N é o número de cromossomos;
2) Calcular o valor objetivo de cada cromossomo da população e armazenar a melhor solução global;
3) Repetir os passos a seguir até que N novos indivíduos sejam criados:
   a) Selecionar um par de cromossomos *"pais"*;
   b) Com uma certa probabilidade, chamada de taxa de *crossover*, recombinar os dois cromossomos em um ponto escolhido aleatoriamente, gerando dois novos indivíduos;
   c) Modificar os dois indivíduos recém-criados com uma certa probabilidade, chamada de taxa de mutação, e adicioná-los à nova população.
4) Substituir a população atual pela nova; e
5) Repetir a partir do Passo 2, caso o valor objetivo desejado não tenha sido atingido.

III. Trabalhos Relacionados

Nesta seção são apresentados trabalhos que tratam de ferramentas criadas para ajudar o game designer no processo de criação de uma jogo. Também são mostrados estudos que visam auxiliar no balanceamento dos jogos pela utilização de algoritmos genéticos.

A. Ferramentas de Auxílio ao Game Designer

Várias tecnologias surgem para facilitar o trabalho de construir jogos eletrônicos. *Engines* são um exemplo claro dessas ferramentas, com motores gráficos, códigos pré-prontos, editor de níveis, dentre outros. Segundo Machado et al. [15], o desenvolvimento de jogos é uma tarefa complexa e que envolve múltiplas camadas e tecnologias. Desenvolvedores e pesquisadores sugerem que assistentes de design de jogos orientados por IA possam melhorar o fluxo de trabalho do desenvolvedor.

Contudo, não apenas *engines* servem para ajudar no processo de desenvolvimento. Machado et al. [15] propuseram um sistema de recomendação para auxiliar no design de jogos ao sugerir mecânicas de jogo que podem ser projetadas de acordo com as características do título que está sendo desenvolvido. Franco et al. [16] propuseram um modelo computacional comportamental para personagens não-controláveis em RPGs eletrônicos, de forma a lhes conferir um comportamentos consistentes com suas personalidades.

Outra tecnologia para auxílio no desenvolvimento de jogos é a Tanagra [17]. Trata-se de uma ferramenta mista para prototipação de níveis de um jogo em plataforma 2D. Sua finalidade é preencher espaços de um nível dentro do jogo, fornecendo uma jogabilidade completa. Caso não seja possível gerar um nível jogável a partir da entrada, o sistema reporta ao projetista que não existe solução viável.

A ferramenta Ropossom [18] também é um gerador de níveis. No entanto, seus objetivos são jogos de quebra-cabeças, em particular o jogo "*Cut the Rope*". O sistema possui dois módulos, um baseado em tecnologias e técnicas de computação evolutiva para gerar os níveis e outro com um agente baseado em algoritmos de busca, que garantem a geração de um nível correto, observando-se que o level design do jogo pode levar a níveis inviáveis.

Algoritmos evolutivos também são a base do trabalho de Liapis et al. [19]. Os autores desenvolveram um procedimento que auxilia o projetista na etapa de criação de mapas. A ferramenta funciona com o designer de jogos dando como entrada esboços de design de níveis para que então, de forma autônoma, a jogabilidade seja verificada e avaliada.

Outra ferramenta que pode ser utilizada para auxiliar na construção de design de jogos é a MaruGen [20]. Tal ferramenta funciona de forma semi-automática e tem como intuito a geração de mecânicas, regras, espaços e missões para os jogos. Seu objetivo é auxiliar na prototipação de jogos, usando como base a definição de regras e a capacidade de explorar os conceitos de progressão do jogo com um método formal. MaruGen é uma ferramenta definida para projetar jogos com elementos inovadores, complexos e comportamentos definidos a partir de combinações de elementos básicos.

B. Balanceamento de Jogos por Meio de AGs

O trabalho de Chen et al. [9] utiliza de ferramentas computacionais, como algoritmo evolucionário e evolução probabilística, como técnica de balanceamento. Seu experimento tem como objetivo equilibrar três personagens com pontuações diferentes de atributos, mas que cheguem em uma mesma quantidade de vitórias batalhando entre si. O teste possui uma construção de *design* simples, que se baseia em três tópicos:

- Existem três raças distintas, com nível máximo L;
- Cada personagem tem apenas uma habilidade e dois atributos que são saúde e taxa de esquiva; e
- Cada habilidade da uma quantidade de dano físico.

Além disso, são definidas as seguintes regras no escopo da busca:

- O nível máximo da classe é 10;
- As taxas de esquiva são $0,2$, $0,6$ e $0,1$ para as classes A, B e C respectivamente;

- O valor de saúde deve obedecer à regra $A > B > C$; e
- A regra de dano deve satisfazer $C > A > B$.

Os resultados reportados por Chen et al. [9] indicam que a utilização de algoritmos evolucionários e evolução probabilística foram capazes de encontrar de forma eficiente o conjunto de personagens proposto no seu escopo. Contudo em seu trabalho se faz necessário a geração de cada nível isoladamente, exemplo: caso um personagem tenha noventa níveis, é necessário encontrar os parâmetros de todos os níveis desejados. Em nossos experimentos, buscamos otimizar de tal forma que seja necessário um menor número de soluções e com uma função de regressão, possamos encontrar todos os níveis desejados.

Outro trabalho que utiliza de métodos evolucionários foi publicado por Morosan e Poli [21]. Segundo os autores, game designers fazem um trabalho de análise de dados e experimentos que demandam tempo para encontrar parâmetros que façam sentido nas tomadas de decisões da IA durante a execução do jogo. Tais parâmetros ideais têm como objetivo construir uma experiência agradável e engajadora para o jogador. Os autores utilizam algoritmos evolucionários com o objetivo de reduzir o tempo gasto nas análises feitas pelos game designers. Morosan e Poli realizaram experimentos aplicados em dois jogos: o clássico *PacMan* e em um sistema mais complexo de um jogo também renomado, *StarCraft*. No experimento com *PacMan*, o objetivo foi definido como tornar o jogo mais fácil para um iniciante. Considerando a taxa de vitória original de 19.67%, desejava-se que o jogador obtivesse 50% das vitórias. Os experimentos obtiveram resultados satisfatórios, encontrando nove parâmetros que correspondiam a simulações que forneciam 50% de vitória para o jogador. Contudo jogos como *PacMan* que disponibilizam um maior número de tomada de decisões é questionável que um jogador se comporte tal qual as simulação feitas para encontrar a porcentagem de vitórias desejadas. Todavia, é questionável assumir que o jogador se comporte tal qual simulado.

Outros autores destacam a importância de trabalhos que colaboram com a agilidade de um processo de design utilizando algoritmos genéticos para encontrar parâmetros ideais. Isso é feito no trabalho de Volz et al. [22], no qual é mostrada a viabilidade de automação do balanceamento dos jogos digitais. Os autores realizam a criação de baralhos de *top trumps*[2] utilizando técnicas de algorítimo genético multiobjetivos. Em seus resultados, foi concluído que há viabilidade do balanceamento automático em termos de qualidade de soluções alcançadas em relação às premissas apresentadas.

O Algoritmo genético é útil, pode ser utilizado em diversas aplicações para encontrar um conjunto de soluções em um escopo definido. Os trabalhos apresentados tiveram suas aplicações em jogos de carta, construção de leveis, recomendação de mecânicas e construção de personagem. Contudo para geração dos personagens deste trabalho, é utilizado apenas de uma modelagem inicial dos monstros e uma taxa de vitória desejada. Também não se faz necessário a geração de todos os possíveis níveis que o personagem precisa ter, graças a utilização da regressão polinomial.

[2]jogo de cartas para crianças, que fez muito sucesso na Inglaterra entre as décadas de 1970 e 1980.

IV. Metodologia

RPG Maker é um motor de jogos para criação de RPG que teve sua origem em 1995. Essa tecnologia já passou por diversas versões e além de ser uma ferramenta com uma alta longevidade, é simples e bastante poderosa até hoje com suas funcionalidades básicas para criação de RPG's em turnos. Decidimos utilizar sua versão mais recente lançada em 2015, o RPG Maker MV.

Com base no modelo de combate da popular *engine* RPG Maker MV. Foi feita também uma implementação própria para a simulação dos combates utilizando a linguagem de programação *Python* em conjunto com o framework de otimização *jMetalPy* [23]. *JMetalPy*[23] facilita a construção de algoritimos por ser um framework baseado em orientação a objetos e sua arquitetura permite facilmente a construção de algortimos evolucionários e definições de objetivos, tanto para problemas com um único objetivo ou multiobjetivos.

A seguir, são descritas as regras utilizadas no modelo da ferramenta e o protocolo experimental proposto para validar o comportamento do balanceamento automático dos atributos do jogador propostos neste artigo.

A. Regras do modelo

1) Representação dos personagens no RPG Maker MV: Os personagens são representados pelos seguintes atributos: ataque (ATQ), ataque mágico (ATM), defesa (DEF), defesa mágica (DFM), *hit points* máximo (HP), *magic points* máximo (MP), agilidade (AGI) e sorte (SOR). Existem ainda outras características em uma entidade: classe, equipamentos e nível para *Player Character* (PC); taxas de erro e esquiva (ESQ), caso de *Non-Player Character* (NPC) que não possui classe que defina.

Note-se que, de acordo com as definições anteriores, um NPC possui características fixas, pois não tem progressão nessa ferramenta. Consequentemente, um inimigo não evoluirá caso derrote o jogador, pois a ferramente não possui um mecanismo para suporte apropriado. Essa regra não é imutável: a ferramenta gera o jogo em *JavaScript* (JS) e a comunidade apresenta uma boa disposição para desenvolvimento de *plugins*, códigos adicionais, normalmente extra-oficiais, que adicionam recursos para o qual o *software* não possuía. Todavia, é importante observar que nem a ferramenta básica nem os jogos padrões não utilizam dessas alterações por se tratar de recurso incomum em RPG.

Por sua vez, em relação aos PC, eles possuem um sistema de progressão definido pela classe, ou seja, cada classe apresenta um conjunto de regras e habilidades pré-determinados durante a progressão dos personagens. Alguns *Digital Role Playing Game* (DRPG) permitem a distribuição de pontos durante a progressão, mas não o RPG Maker MV. Essa ferramenta estabelece quais as habilidades serão adquiridas e como os

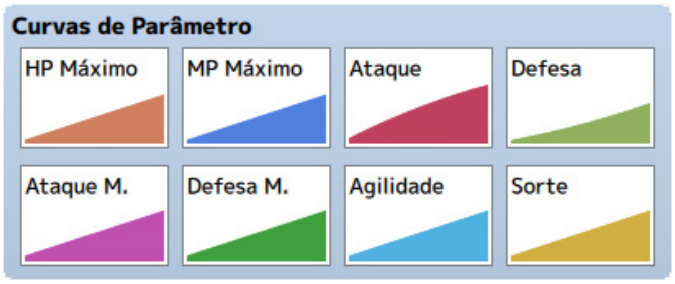

Fig. 2. Curvas de progressão padrões da classe "herói" na ferramenta RPG Maker MV, as quais apresentam um crescimento linear. Fonte: adaptado da ferramenta RPG Maker MV pelos autores.

pontos dos personagens serão aumentados à medida que o PC progride de nível de acordo com a sua classe.

Além de definir equipamentos possíveis e habilidades, a classe também determina a progressão dos personagens e seus atributos. Por exemplo, as curvas que regem o crescimento da classe "herói" são ilustradas na Fig. 2. No entanto, essa progressão não é completa, pois ela ainda é influenciada pelas habilidades do personagem e, principalmente, por seus equipamentos. Além disso, as curvas padronizadas no RPG Maker MV são majoritariamente lineares e com crescimento constante, o que pode levar a um desbalanceamento, gerando personagens muito fracos ou muito fortes se não houver o devido tratamento por parte do game designer. De maneira geral, deseja-se que o crescimento dos equipamentos acompanhe a progressão dos atributos.

2) Algoritmo de Combate: O RPG Maker MV possui uma estrutura e um algoritmo bem definidos para a execução de uma batalha. Todavia, as manobras realizadas pelos combatentes são determinadas por habilidades customizáveis, ou seja, é natural que cada jogo produzido possua suas peculiaridades de combate. Neste trabalho, visando maior aplicabilidade, é utilizado como base o modelo apresentado como padrão da ferramenta.

Os atributos envolvidos são customizáveis e costumam serem alteráveis por outros fatores, tais como habilidades escolhidas na manobra, equipamentos e outros efeitos temporários. O processo da batalha pode variar de acordo com o uso de habilidades diversas, contra diferentes inimigos e seus efeitos. O processo a seguir é resumido assumindo apenas ataques básicos entre os oponentes:

1) **[SELECIONAR AÇÃO]:** Cada personagem escolhe uma ação e o seu alvo, que é opcional dependendo da ação;
2) **[DEFINIR ORDEM]:** Cálculo da iniciativa, que é dado sob um valor básico que inclui habilidade, arma e algum outro bônus possível;
3) **[ERRO E ESQUIVA]:** O atacante pode errar o alvo. Assim, o sucesso é calculado sobre uma taxa de acerto definida por uma especificação prévia da classe, da arma e da habilidade usada. Caso o atacante erre, o processo é encerrado. Caso ele não erre, o alvo ainda pode se esquivar, sendo que sua taxa de esquiva é pré-definida pela classe e não sofre influência de outro personagem. Da mesma forma que o teste anterior, caso o alvo se esquive, o ataque é completamente ignorado;
4) **[CÁLCULO DO DANO]:** Esta etapa pode variar mais que as anteriores. Neste trabalho são utilizadas somente as habilidades básicas de ataque (ATQ) e de defesa (DEF), porém a abordagem pode ser adaptada para lidar com outras manobras. O cálculo do dano é mostrado em (1).

$$\text{DANO} = (4 \cdot \text{C}_1.\text{ATQ} - 2 \cdot \text{C}_2.\text{DEF}) \cdot \text{VAR} \quad (1)$$

VAR é a taxa de variação, um valor aleatório dentro de um intervalo específico. Por exemplo, a taxa de variação padrão é de 20%, de modo que esse valor deve variar dentro do intervalo I de $0,8$ a $1,2$.
5) **[CÁLCULO DE CRÍTICO]:** Após o ataque, a habilidade passa por um teste de crítico. A manobra padrão de ataque do RPG Maker MV possui esta opção habilitada. O teste de crítico leva em consideração a definição de habilidade e da classe. Se o teste for bem sucedido, o valor obtido de ATQ em (1) é triplicado. Caso contrário, seu valor não é alterado; e
6) **[RETORNE AO COMEÇO OU ENCERRE O COMBATE]:** O processo continua até que todas as ações sejam esgotadas ou os personagens que iriam executá-las ainda não tenham perecido. No final da rodada, caso um dos lados ainda não tenha vencido o processo, retorna-se à etapa 1 da lista, [SELECIONAR AÇÃO], e o repete-se o processo.

A habilidade de atacar apresentada em (1) funciona como uma variável e poderia ser substituída por outros valores e reavaliada nas simulações. Dessa forma, os atributos que envolvem habilidades mágicas (MP, ATM e DFM) e a Sorte não sofreram alterações no experimento, pois não são utilizadas no algoritmo de combate padrão.

B. Montagem do Experimento

1) Simulação: A simulação é a implementação direta do modelo básico padrão da ferramenta RPG Maker MV. Cada combatente possui seis atributos: HP, ATQ, DEF, AGI, ESQ e CRI. A simulação recebe como entrada dois conjuntos de atributos, um para o jogador e outro para o seu oponente. Os valores recebidos são transformados em inteiros, pois a *engine* não aceita valores reais. A simulação ocorre ocorre de acordo com os passos mostrados na Fig. 3, até que o HP de um dos combatentes chegue a zero.

Uma simulação dá um resultado binário: 1 se o jogador, 0 caso contrário. No entanto, devido aos cálculos probabilísticos envolvidos, como o cálculo de acerto crítico e esquiva, esse resultado não é determinístico. Desse modo, executar uma simulação com os mesmos valores de atributos tanto para o jogador quanto para seu inimigo pode dar um resultado final diferente. Assim, para obter uma porcentagem de vitórias do jogador confiável, é necessário executar a simulação diversas vezes. Neste trabalho, foram executadas 10 mil simulações.

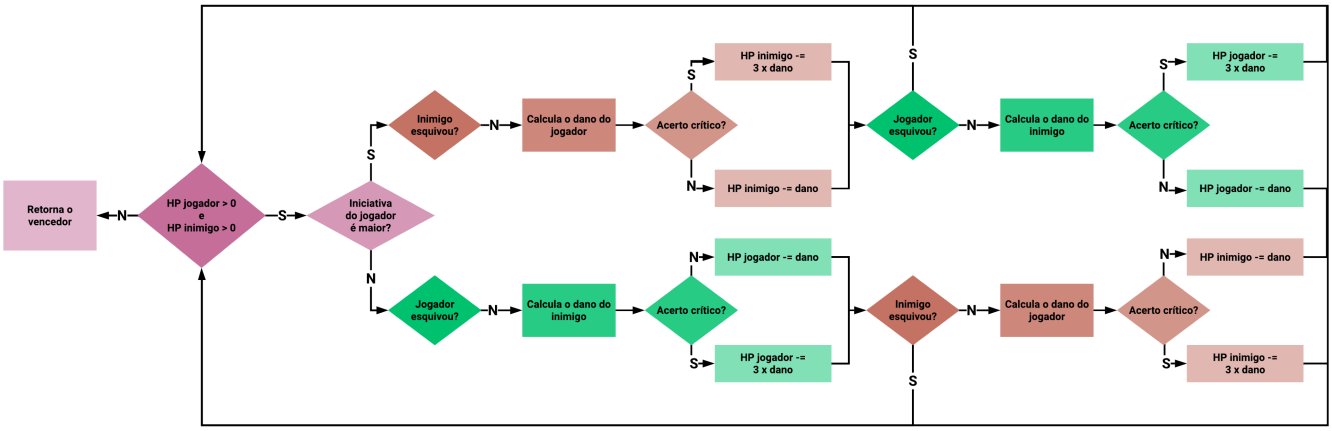

Fig. 3. Sequência de passos da simulação de batalha. Os fluxos dos combatentes são simétricos, porém os passos jogador estão destacados em verde. Note-se que essa simulação poderia ser substituída por outro fluxo, em vias de adaptar a abordagem proposta para outros sistemas e situações de batalha. Fonte: os autores.

2) Objetivo: A função de objetivo utilizada é a minimização do erro entre a taxa de vitória desejada, tax_{des}, e a taxa de vitória obtida através da simulação, tax_{sim}. Observe que se o jogador sempre perder, o erro será sempre igual a taxa desejada, o que pode indicar erroneamente um valor ruim como bom. Por exemplo, se $tax_{des} = 0,25$ e o o jogador sempre perder, o erro será sempre $0,25$. Note que um jogador que vença 51% dos combates terá um erro de $0,26$. Assim, o algoritmo diria que perder sempre é um resultado melhor do que perder 49% das vezes. Isso não é correto, pois no último caso houve alguma movimentação dos parâmetros em busca de uma solução. Já no primeiro caso, somente valores muito ruins darão uma derrota em todas as simulações. Da mesma forma, acontecerá um problema análogo se o jogador sempre vencer.

Para corrigir esse problema, caso o jogador sempre perca, ou seja $tax_{sim} = 0$, ou sempre vença, ou seja $tax_{sim} = 1$, o erro será automaticamente máximo. O cálculo do erro é mostrado em (2).

$$\text{Erro} = \begin{cases} 1, & \text{se tax}_{sim} = 0 \text{ ou } 1 \\ |\text{tax}_{sim} - \text{tax}_{des}|, & \text{c.c.} \end{cases} \quad (2)$$

Para o cálculo da taxa de simulação utilizada na função objetivo, a simulação foi executada por 10 mil vezes.

Uma consideração importante é que caso os valores iniciais fossem muito distantes de uma solução, o algoritmo precisaria de muitas iterações até encontrar um resultado dentro da margem de erro desejada. Dessa forma, foi utilizado um critério de parada baseado no valor objetivo que avalia as soluções obtidas e só para quando há pelo menos um cromossomo cujo resultado está dentro da margem de erro. Apesar desse indicador demorar mais para dar algum resultado, há a garantia de que uma vez que haja uma solução, ela será adequada.

3) Restrições: Um dos fatores a ser considerado na geração de uma curva de nível é a progressão crescente dos atributos, ou seja, que os valores de um nível atual sejam iguais ou maiores do que os do nível anterior. De maneira geral, não se espera que uma habilidade do jogador diminua após ele subir de nível. Sendo assim, é preciso garantir que o algoritmo gere valores coerentes com o objetivo. A maneira utilizada neste trabalho foi definir um conjunto de restrições de valores mínimos e máximos para cada nível do jogador.

As duas restrições básicas são que o jogador não pode ter nenhum atributo com valor menor do que zero ou maior do que o máximo permitido pelo RPG Maker MV. Para os atributo HP, o valor máximo é de 9.999. Para todos os outros, o máximo é 999. Entretanto, somente essas duas restrições não garantem uma progressão crescente dos atributos, para isso é necessário utilizar uma regra mais complexa.

Para o primeiro nível, o jogador deverá ter valores maiores ou iguais à metade dos valores do inimigo. A partir do nível 2, os atributos do jogador nunca poderão ser menores do que os do nível anterior. Já os valores máximos até o penúltimo nível nunca devem ser maiores do que os do inimigo do nível seguinte. No último nível, os valores máximos são os básicos, descritos anteriormente. Observe que para garantir uma curva de nível sempre crescente, é necessário que os atributos dos inimigos definidos pelo game designer também sigam essa regra.

4) Hiperparâmetros: Por uma questão de reprodutibilidade dos nossos resultados, todos os hiperparâmetros utilizados no algoritmo genético da solução jMetalPy são listados a seguir. O tamanho da população usado foi de 100 cromossomos. O operador de cruzamento utilizado foi o SBX [24], chamado de *SBXCrossover* no jMetalPy, com probabilidade de $0,9$ e índice de distribuição 20. Foi usada uma mutação polinomial [25], *PolynomialMutation* no jMetalPy, com probabilidade de $0,167$, ou $1/(\text{NV})$, onde NV é número de variáveis e índice de distribuição 20. Para a etapa de seleção foi utilizado um seletor binário, *BinaryTournamentSelection* no jMetalPy. Por fim, o critério de parada utilizado seguiu o indicador de

qualidade descrito anteriormente, *StoppingByQualityIndicator* no jMetalPy, com uma margem de erro de $0,05$.

C. Protocolo Experimental

As etapas do protocolo experimental proposto são apresentadas a seguir (vide Fig. 1 para um resumo). No diagrama há uma representação visual do processo desde a definição dos valores iniciais até a etapa de geração de curvas de nível.

1) Definição dos valores iniciais: Esta é a única etapa que precisa ser realizada pelo *game designer*. Previamente, o game designer deve preparar um conjunto de valores de atributos para cada nível do inimigo de referência. Note que quanto mais níveis forem definidos, maior será a granularidade gerada pelo algoritmo de balanceamento automático. Assim como no RPG Maker MV, podem ser utilizados níveis de 1 a 100. Esses valores devem seguir as mesmas restrições descritas anteriormente.

Também devem ser definidas pelo game designer as taxas de vitória desejadas por nível. Por ser uma porcentagem, são aceitos valores reais de 0 a 1. Cada nível pode possuir uma taxa de vitória associada, mas o game designer tem a liberdade, por exemplo, de manter uma taxa igual em todos eles.

2) Balanceamento dos atributos: Uma vez com os valores iniciais em mão, o game designer poderá passá-los à ferramenta de balanceamento automático. Ela cuidará da geração automática dos atributos para cada um dos níveis, utilizando um algoritmo genético. Ao final do processo, é retornada uma lista de atributos do jogador correspondentes à taxa de vitória desejada contra o inimigo para cada nível.

3) Geração das Curvas de Nível: Com os valores retornados, são geradas as curvas de nível para cada um dos atributos. Nesse ponto, o game designer já terá todos os valores de atributos desejados sem ter que realizar o balanceamento manualmente. No entanto, ainda não há uma curva propriamente dita. Mais do que isso, os valores brutos possuem variações bruscas, as quais não condizem com uma progressão suave dos atributos. Assim, a curva final utilizada é gerada com base nos valores retornados pela ferramenta de balanceamento após passar por um algoritmo de regressão polinomial. Com isso, o crescimento se mantém, mas com uma progressão mais suave. No fim, são esses valores filtrados que são entregues como resultado final do processo de balanceamento automático.

4) Exportação das Curvas de Nível para o RPG Maker MV: Por fim, os valores obtidos através da ferramenta de balanceamento podem ser exportados em formato *.json*. Esse formato permite guardar os valores de uma maneira semiestruturada, contendo os valores de cada um dos atributos. Além disso, esse é um formato aceito pelo RPG Maker MV, de forma que as curvas possam ser importadas diretamente na *engine*, sem a necessidade de mais um processo manual.

V. Resultados e discussão

Para avaliar a eficácia da ferramenta de balanceamento automático, foi desenvolvido um caso experimental. Foi definido previamente um inimigo com dez níveis diferentes, cada um com seus respectivos atributos. Cada nível corresponde a um nível do jogador e a porcentagem de vitória desejada na batalha entre eles foi definida como 80%, com margem de erro de 5% para mais ou para menos. Os valores definidos são apresentados na Tabela I. Em um jogo real, esses valores seriam definidos pelo game designer, independentemente de utilizar ou não uma estratégia de otimização como a proposta.

TABLE I
EXEMPLO DE TABELA DE NÍVEIS CRIADA PELO GAME DESIGNER

Nível	HP	ATQ	DEF	AGI	ESQ	CRI
1	100	20	35	0	0	0
10	987	79	57	5	1	2
20	1.996	103	100	10	2	4
30	2.800	158	170	12	4	6
40	4.155	200	200	15	6	9
50	4.700	272	262	23	7	10
60	5.153	317	320	31	10	13
70	6.239	379	430	45	13	16
80	7.356	478	503	50	15	20
90	8.980	578	600	64	18	27

Definida a tabela, a lista de atributos por nível e as taxas de vitória desejadas são passadas como entrada à ferramenta. O processo de otimização é executado para cada nível e retorna a lista de atributos balanceados. Um exemplo de resultado de balanceamento é mostrado na Tabela II. Esse exemplo resultou nas taxas de vitória apresentadas na Tabela III após 10 mil simulações em cada um dos níveis, em que todas as taxas estão dentro da margem de erro da porcentagem desejada $[75\%, 85\%]$. Note que devido à natureza estocástica do algoritmo genético, os resultados serão ligeiramente diferentes a cada execução. Dessa forma, é possível ver que o resultado gerado no processo de balanceamento foi adequado ao objetivo proposto.

TABLE II
EXEMPLO DE RESULTADO GERADO APÓS O BALANCEAMENTO AUTOMÁTICO

Nível	HP	ATQ	DEF	AGI	ESQ	CRI
1	64	41	22	1	0	0
10	1.722	58	64	9	0	1
20	2.268	118	81	10	2	3
30	2.673	135	238	10	4	4
40	4.652	185	242	22	6	6
50	5.015	313	244	24	9	10
60	5.869	340	338	31	9	12
70	6.059	404	510	43	11	12
80	8.192	457	611	49	15	24
90	8.770	520	722	84	27	43

Ao fim do processo, tendo todos os valores para cada um dos níveis desejados, será criada uma curva de nível para o jogador. Assim, o game designer terá o valor de cada um dos atributos do jogador para cada nível em que ele se encontra sem ter que defini-los manualmente. Observe na Tabela II que apesar de seguir todas as restrições, os resultados a princípio não seguem uma curva muito suave. Por exemplo, o HP do jogador dá um salto de 64 para 1.722 do nível 1 ao 10. Ou ainda, a agilidade se mantém em 10 nos níveis 20 e 30, mas aumenta consideravelmente de 49 a 84 nos níveis 80 e 90. Esse fato pode ser observado mais facilmente através das curvas

TABLE III
TAXAS DE VITÓRIA OBTIDAS NA SIMULAÇÃO COM AS RESPOSTAS GERADAS

Nível	Taxa de vitória	% de erro
1	76,61%	3,39%
10	82,86%	2,86%
20	81,44%	1,44%
30	80,11%	0,11%
40	78,80%	1,20%
50	80,58%	0,58%
60	79,59%	0,41%
70	80,26%	0,26%
80	80,89%	0,89%
90	80,35%	0,35%

de nível preliminares apresentadas na Fig. 4. As curvas de nível resultantes são suavizados por uma função de regressão polinomial para produzir as curvas de nível finais, mostradas na Fig. 5.

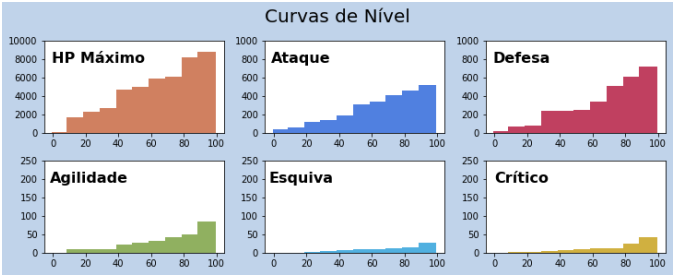

Fig. 4. Curvas de progressão preliminares geradas pelo algoritmo genético. Imagem ilustrativa. Fonte: os autores

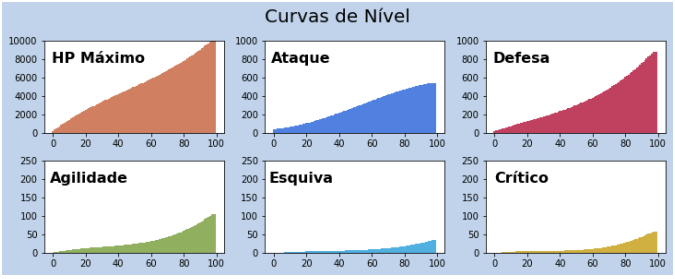

Fig. 5. Curvas de progressão suavizadas após a execução de uma regressão polinomial. Imagem ilustrativa. Fonte: os autores

É importante ressaltar que esse foi um exemplo específico. Caso o game designer deseje, ele poderia definir valores de todos os 100 níveis do inimigo. Observe que a solução aqui apresentada independe da equação matemática definida no **CÁLCULO DE DANO** definido na Seção IV-A. Vencida essa parte de definição do desenvolvimento do *game design*, e implementado um sistema que pelo menos simule as batalhas propostas no jogo, o designer ou a equipe responsável pelo sistema de progressão do herói deixa de se preocupar com as curvas apresentadas na Fig. 2 e passa a se preocupar somente com a definição dos inimigos e as taxas de vitória.

VI. CONCLUSÃO

Os jogos de RPG geralmente demandam a construção de curvas de progressão dos personagens e das criaturas. A geração e o balanceamento dessas curvas por muitas vezes é feita manualmente com base em tentativa e erro. Contudo, caso a progressão dos personagens não apresente a experiência que o game designer gostaria, podem ocorrer frustrações ou tédio. Esse artigo investigou técnicas de inteligência artificial que podem ajudar no desenvolvimento de curvas de progressão com base nos inimigos já existentes. Nesse trabalho, foi apresentado um processo de balanceamento automático de atributos do jogador para geração de curvas de níveis de RPG.

A técnica apresentada baseia-se no uso de otimização via algoritmo genético para que o game designer não precise se preocupar em quais valores de parâmetros o jogador deve possui e se concentrar somente na modelagem das criaturas e no percentual de vitória que deseja para o jogador nessas batalhas. Os experimentos foram conduzidos considerando que estão definidos previamente dez níveis de um inimigo hipotético e que o personagem deveria ter uma taxa de 80% de vitória, com uma margem de erro de no máximo 5%. Como visto na Tabela III, para cada um os dez níveis utilizados como exemplo na modelagem, a ferramente gerou valores adequados, com uma taxa de erro máxima de 3.39%, que respeita o limite imposto. As curvas de nível suavizadas resultantes podem então ser passadas à ferramenta usada no estudo, RPG Maker MV.

Note-se que, apesar de não ter sido validada com essa finalidade, a abordagem proposta é flexível o suficiente para ser aplicada de modo a ajustar a dificuldade dos inimigos à progressão do jogador, a fim de conferir maior valor de rejogabilidade a um produto. Por fim, podemos citar os principais trabalhos futuros: a técnica proposta pode ser integrada com outras ferramentas para criação de sistemas de combate de RPGs, como a *machination*[3]; detecção de pontos fora das curvas, que representariam desbalanceamentos nos resultados e a subsequente retroalimentação desses atributos na função objetivo, de modo a evitar esse tipo de problema; e um componente de simulação de batalha poderia explorar possibilidades mais complexas, com diferentes sistemas, grupos de combatentes, novas ações e estratégias gerais.

REFERENCES

[1] A. O. Franco, T. V. Rolim, A. M. Santos, J. W. Silva, V. M. Vidal, F. A. Gomes, M. F. Castro, and J. G. Maia, "An ontology for role playing games," *Proceedings of SBGames*, pp. 615–618, 2018.

[2] A. M. M. Santos, A. Franco, J. G. R. Maia, F. Gomes, and M. Castro, "A methodology proposal for mmorpg content expansion analysis," in *16th Brazilian Symposium on Computer Games and Digital Entertainment*. SBC, 2017, pp. 115–124.

[3] J. Schell, *The Art of Game Design: A book of lenses*, 3rd ed. AK Peters/CRC Press, 2019.

[4] C. Crawford, *The art of computer game design*. Osborne/McGraw-Hill Berkeley, CA, 1984.

[3]https://machinations.io

[5] D. M. Maranhão, G. M. M. Junior, A. d. O. da Rocha Franco, and J. G. R. Maia, "Towards a comprehensive model for analysis and definition of game mechanics," in *15th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. SBC, 2016, pp. 581–590.

[6] M. T. Islam, K. M. Nahiduzzaman, Y. P. Why, and G. Ashraf, "Learning character design from experts and laymen," in *International Conference on Cyberworlds*. IEEE, 2010, pp. 134–141.

[7] F. G. Faria, L. T. Pereira, and C. F. M. Toledo, "Adaptive generation of characters for tabletop role playing games," in *18th Brazilian Symposium on Computer Games and Digital Entertainment*. IEEE, 2019, pp. 39–46.

[8] J. A. Brown, "Evolved weapons for rpg drop systems," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, 2013, pp. 1–2.

[9] H. Chen, Y. Mori, and I. Matsuba, "Solving the balance problem of online role-playing games using evolutionary algorithms," *Journal of Software Engineering and Applications*, 2012.

[10] E. S. de Lima, B. Feijó, and A. L. Furtado, "Procedural generation of quests for games using genetic algorithms and automated planning," in *18th Brazilian Symposium on Computer Games and Digital Entertainment*. IEEE, 2019, pp. 144–153.

[11] T. Mantere and J. Koljonen, "Solving and rating sudoku puzzles with genetic algorithms," in *New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP*, 2006, pp. 86–92.

[12] D. Goldberg, "Genetic algorithms in search, optimization, and machine learning.(1989) addisonwesley," *Reading, Mass*, 1989.

[13] M. A. C. Pacheco *et al.*, "Algoritmos genéticos: princípios e aplicações," *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, p. 28, 1999.

[14] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[15] T. Machado, D. Gopstein, A. Wang, O. Nov, A. Nealen, and J. Togelius, "Evaluation of a recommender system for assisting novice game designers," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, no. 1, 2019, pp. 167–173.

[16] A. O. R. Franco, J. G. R. Maia, J. A. M. Neto, and F. A. C. Gomes, "An interactive storytelling model for non-player characters on electronic rpgs," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2015, pp. 52–60.

[17] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: A mixed-initiative level design tool," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 2010, pp. 209–216.

[18] N. Shaker, M. Shaker, and J. Togelius, "Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels," in *9th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[19] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: computer-assisted game level authoring," in *FDG*. ACM, 2013.

[20] I. F. Capasso-Ballesteros and F. De la Rosa-Rosero, "Semi-automatic construction of video game design prototypes with marugen," *Revista Facultad de Ingeniería Universidad de Antioquia*, 2020.

[21] M. Morosan and R. Poli, "Automated game balancing in ms pacman and starcraft using evolutionary algorithms," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 377–392.

[22] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 269–276.

[23] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. D. Ser, "jmetalpy: A python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, p. 100598, 2019.

[24] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.

[25] K. Deb and S. Tiwari, "Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1062–1087, 2008.