NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AUTOMATED FLAT CIRCUIT-LEVEL TOPOLOGY GENERATION

CIRCUIT SYNTHESIS USING GENETIC ALGORITHMS

MIGUEL PINTO CAMPILHO GOMES

Master/BSc in Electrotechnical and Computer Engineering

DOCTORATE IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
November, 2024

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AUTOMATED FLAT CIRCUIT-LEVEL TOPOLOGY GENERATION

CIRCUIT SYNTHESIS USING GENETIC ALGORITHMS

MIGUEL PINTO CAMPILHO GOMES

Master/BSc in Electrotechnical and Computer Engineering

Adviser: Rui Manuel Leitão Tavares
Assistant Professor, NOVA University of Lisbon

Co-adviser: João Carlos da Palma Goes
Full Professor, NOVA University of Lisbon

Examination Committee

Chair: José António Barata de Oliveira
Full Professor, NOVA University of Lisbon

Rapporteurs: Nuno Cavaco Gomes Horta
Full Professor, Instituto Superior Técnico, University of Lisbon

Jorge Manuel Correia Guilherme
Adjunct Professor, Polytechnic Institute of Tomar

Adviser: Rui Manuel Leitão Tavares
Assistant Professor, NOVA University of Lisbon

Members: José António Barata de Oliveira
Full Professor, NOVA University of Lisbon

Nuno Filipe Silva Veríssimo Paulino
Associate Professor with Habilitation, NOVA University of Lisbon

DOCTORATE IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
November, 2024

Automated Flat Circuit-Level Topology Generation
Circuit Synthesis using Genetic Algorithms

Parts of this document were created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.29) [11].

Acknowledgments

First of all, I would like to thank my supervisors for all their support throughout the journey of this work, which was essential for it to come to a fruition. Prof. Rui Tavares and Prof. João Goes stood firm in the conviction that the line of research pursued in this thesis would yield valuable results, and this conviction was a supporting pillar to keep my research efforts going.

To my colleagues at CEDET[1], I would like to say a very special and heartfelt thank you for all that you have contributed to my well-being, both professionally and otherwise, which has been crucial in getting this work done and seeing it through to the end.

I would also like to express my sincere gratitude to Prof. Pedro Santos, with whom I worked at IT-Lisboa[2] and from whom I and learnt a lot about layout design of integrated circuits; and to whom I am indebted as it was thanks to his positive and motivating challenge that this endeavour was originally set in motion.

A number of institutions have also contributed to this work, either indirectly or directly, and I would like to acknowledge their contribution as well. This work was supported by INCD[3] funded by FCT[4] and FEDER[5] under project 01/SAICT/2016 nº 022153 and was financed by FCT/MCTES in the scope of UIDB/00066/2020 (PEST) and PTDC/CTM-PAM/4241/2020 (IDS-PAPER) projects. This work also had the financial support from FCT within the scope of CTS[6] multiannual strategic funding[7], and in the context of IDS-Paper R&D project[8].

[1] Centro de Estudos e Desenvolvimento de Eletrónica e Telecomunicações do ISEL.
[2] Instituto de Telecomunicações, polo de Lisboa.
[3] Infraestrutura Nacional de Computação Distribuída.
[4] Fundação para a Ciência e a Tecnologia.
[5] Fundo Europeu de Desenvolvimento Regional.
[6] Center for Technology and Systems.
[7] UIDB/00066/2020 and UIDP/00066/2020
[8] PTDC/CTM-PAM/4241/2020

I would also like to thank ISEL[9] for the resources made available.

Last but not least, I would like to thank my family for their constant support over the years and apologize for all the time I have put into this work that has not been spent with them. In particular, I would like to apologize to Cristina, João and Guilherme, as well as to my brother Nuno and my parents, Alberto and Regina (my late mother). I wish I could have been there more.

[9]Instituto Superior de Engenharia de Lisboa.

Abstract

Digital and analog circuit design for integrated circuits classically has three distinct stages: topology choice, component sizing and layout generation. In the Electronic Design Automation (EDA) field, the automatic topology synthesis stage is the least developed stage, particularly for analog circuits. Analog circuit design is still essentially based on human expert designers and is generally geared toward the reuse of well-known topologies in new circuit functions or to improve existing circuit topologies and, less often, to create new circuit topologies.

Generation of circuit topologies can be seen as a search and optimization problem and, therefore, it can be performed with Genetic Algorithms (GAs). What is presently unclear is to what extent this can be accomplished by a GA, particularly when handled at flat circuit-level, and in concrete for the case of MOS integrated amplifiers. Boosted by the current submicron technologies used in today's integrated circuits, MOS integrated amplifiers have high efficiencies and very optimized specifications compared to the not-so-distant past. Eventually, an automatically generated topology could not only meet such specifications, but achieve it being a novelty, that is, potentially a GA can generate new topologies different from the traditional ones.

The work described in this thesis concerns the use of GAs in automatic circuit synthesis at flat circuit-level, and tries to understand its merits and its limitations, *i.e.*, what are the capabilities of a GA in producing useful circuits and what is the relevance of additional enhancement techniques in its overall performance. In this work a GA kernel has been developed using parallelism in some stages of the algorithm, namely where it is most relevant which is in the fitness evaluation stage. In this stage, fitness evaluation is accomplished by a spice-like circuit simulator run using thread-level parallelism.

Part of the difficulty of obtaining good results with GAs is the choice of the codification scheme of the problem. Several coding schemes can be used for electric circuits and in

this work there was a first attempt to use a codification technique for circuits that applies the classic bit strings used by the canonic (or original) GA to represent a circuit. In this representation there is a fixed number of components available to the GA and the evolution takes place by changing the nodes to which each component terminal is connected to. In several experimental results obtained with this coding simple circuits were synthesized, such as NAND gates with MOS transistors. But scaling to more complex topologies was not possible. Hence, another codification technique was attempted, and variable-length chromosomes (VLCs) were tested.

Using VLCs and an encoding scheme in which a chromosome is a circuit descriptor and each gene is a component descriptor, circuits of higher complexity were synthesized by the GA, like other logic gates never successfully generated before (*e.g.*, XOR gate and half-adder) and analog circuits (such as amplifiers of gain up to 40 dB and $GBW \approx 70\,\text{MHz}$). In addition, a few techniques have been developed and incorporated in the GA which contribute to its robustness, leading to better and faster results. Robustness is enhanced because the GA has less sensitivity to initial pseudo-random generator seed and to some parameters. Better results are obtained since circuits are produced with less redundant or useless components. Also, faster results are obtained because less time is spent in circuit simulation (since the number of components in intermediate circuits is high only during a set of iterations strictly necessary).

One enhancement technique presented is a heuristic that implements an adaptive probability of acceptance of mutated chromosomes according to a measure of the quality of the solutions obtained. This technique enhances local exploration capabilities of the GA and reduces sensitivity to mutation probability. Another improvement technique is a "circuit cleaning" technique that eliminates redundant components in the circuit in which we dynamically adapt the weight of a constraint to steer the algorithm towards the sub-objective of minimizing the number of components. In addition to this technique, a segmentation procedure of the GA evolution was presented, where several phases are used, each with different parametrization, which is a valuable technique to extend the steering of the GA towards sub-objectives. Circuits synthesized by the GA were shown using all the techniques described.

Resumo

O projeto de circuitos digitais ou analógicos para circuito integrado tem classicamente vários estágios distintos: a escolha da topologia, o dimensionamento dos componentes e a geração do *layout*. Na área de "Electronic Design Automation", o estágio da síntese automática da topologia é o menos desenvolvido, principalmente para o caso dos circuitos analógicos. O projeto de circuitos analógicos ainda é essencialmente baseado em projetistas (humanos) especializados, sendo muito voltado para a reutilização de topologias conhecidas em novas funções ou para o melhoramento das topologias já existentes, sendo com menos frequência orientado para a criação de novas topologias.

A geração de topologias de circuitos pode ser vista como um problema de pesquisa e otimização e, portanto, pode ser realizada com Algoritmos Genéticos (AGs). O que não é claro atualmente é até que ponto isto pode ser realizado por um AG, particularmente com síntese ao nível do componente e não de subcircuitos ou de topologias preexistentes, e em concreto para o caso de amplificadores integrados MOS.

Impulsionados pelas tecnologias submicrométricas atualmente usadas no fabrico de circuitos integrados, os amplificadores integrados MOS têm hoje elevada eficiência e especificações muito otimizadas em comparação com um passado não muito distante. Eventualmente, uma topologia gerada automaticamente por um AG poderia não apenas conseguir cumprir as mesmas especificações, ou mesmo melhorá-las, e poderia até constituir uma novidade, ou seja, potencialmente um AG pode gerar novas topologias.

O trabalho descrito nesta tese é sobre o uso de AGs na síntese automática de circuitos ao nível do componente. Nele tenta-se conhecer os méritos e as limitações dos AGs neste contexto, ou seja, tenta-se conhecer quais são as capacidades de um AG de produzir circuitos úteis e qual a relevância de algumas técnicas adicionais de melhoramento, desenvolvidas neste trabalho, no seu desempenho geral. Para este efeito foi desenvolvido um programa que realiza um AG, utilizando paralelismo em alguns estágios do algoritmo, nomeadamente

onde é mais relevante que é no estágio de avaliação de desempenho de cada circuito. Neste estágio, a avaliação de desempenho é realizada por um simulador de circuitos baseado em SPICE, executado usando paralelismo ao nível da *thread*.

Parte da dificuldade na obtenção de bons resultados com AGs está na escolha da codificação do problema. Várias formas de codificação podem ser usadas para a síntese de circuitos elétricos e neste trabalho houve uma primeira tentativa de se usar uma técnica de codificação que aplica as sequências de bits clássicas usadas no AG canónico (ou original). Nesta representação existe um número fixo de componentes disponíveis para o AG realizar o circuito e a evolução ocorre por alteração dos nós aos quais cada terminal de um componente está conectado. Em vários resultados experimentais obtidos com esta codificação foram sintetizados circuitos simples, como portas NAND com transístores MOS. Mas o escalamento para topologias mais complexas não foi conseguido. Consequentemente, foi tentada outra técnica de codificação, tendo sido escolhida e depois testada a utilização de cromossomas de tamanho variável (CTVs).

Usando CTVs e um esquema de codificação em que cada cromossoma é um descritor de circuito e cada gene é um descritor de componente, foram sintetizados com sucesso pelo AG circuitos de maior complexidade, tais como portas lógicas nunca antes geradas com sucesso (por exemplo, uma porta XOR e um *half-adder* de duas entradas) e circuitos analógicos (como amplificadores de ganho até 40 dB e $GBW \approx 70\,\text{MHz}$). Além disso, algumas técnicas foram desenvolvidas e incorporadas no AG, contribuindo para a sua robustez e levando a resultados melhores e mais rápidos. A robustez é melhorada porque o AG fica com menos sensibilidade não só à semente inicial do gerador pseudo-aleatório mas também a alguns parâmetros do próprio AG. Como consequência são obtidos melhores resultados porque os circuitos são produzidos com menos componentes redundantes ou inúteis. Acresce ainda que passa a haver maior rapidez na obtenção de resultados porque o tempo total gasto na simulação de circuitos diminui (uma vez que o número de componentes existente nos circuitos intermédios é em média menor, sendo elevado apenas durante um conjunto de iterações em que tal é estritamente necessário).

Uma técnica de melhoramento de desempenho do AG apresentada neste trabalho é uma heurística que implementa uma probabilidade adaptativa de aceitação de cromossomas mutados de acordo com uma medida da qualidade das soluções obtidas. Esta técnica aumenta as capacidades de exploração local do AG e reduz a sensibilidade à probabilidade de mutação. Outra técnica de melhoramento é uma técnica de "limpeza de circuito" que elimina componentes redundantes ou inúteis no circuito, na qual se adapta dinamicamente o peso de uma restrição para se direcionar o algoritmo para o subobjetivo de minimizar o número de componentes usados no circuito. Além desta técnica, foi apresentado um procedimento de segmentação da evolução do AG onde são utilizadas várias fases, com parametrizações diferentes, o que constitui uma técnica que permite direcionar o AG para vários sub-objetivos. Usando estas técnicas, descritas e desenvolvidas neste trabalho, vários circuitos sintetizados pelo AG foram apresentados e discutidos nesta tese.

Contents

List of Abbreviations and Acronyms

A

A&MS Analog and Mixed-Signal
ADA Analog Design Automation
AI artificial intelligence
AOL open-loop gain
ASIC Application Specific Integrated Circuit

B

BJT bipolar junction transistor

C

CAD Computer-Aided-Design
CMRR Common-mode Rejection Ratio

D

DE Differential Evolution

E

EA Evolutionary Algorithm
EDA Electronic Design Automation

F

FLC fixed-length chromosome
FoM Figure of Merit
FPAA Field Programmable Analog Array

FPGA Field Programmable Logic Array
FPTA Field Programmable Transistor Array

G

GA Genetic Algorithm
GA-c canonical or conventional Genetic Algorithm
GBW Gain-bandwidth product
GP Genetic Programming

I

IC Integrated Circuit
IoT Internet of Things
ITRS International Technology Roadmap for Semiconductors

K

KPI key performance indicator

L

LPF low-pass filter

M

MOEA Multi-Objective Evolutionary Algorithm
MOOEA Multi-Objective Optimization Evolutionary Algorithm
MS-SoC Mixed-Signal System-on-a-Chip

O

ONCR Ordered Node Clustering Representation
opamp operational amplifier
OS Output voltage swing

P

PCB Printed Circuit Board
PLD Field Programmable Device
PM phase margin
PSRR Power Supply Rejection Ratio

S

SoC System-on-a-Chip
SR Slew-rate

T

THD Total Harmonic Distortion

V

VEGA Vector-Evaluated Genetic Algorithm
VLC variable-length chromosome
VLR Variable-length Representation

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter introduces the motivation and main objectives of this work. It also presents the main research question and its hypothesis.

1.1 Background and motivation

Currently, analog circuit design is still essentially based on specialized designers and is often considered an art [Williams, 1998, Ferent and Doboli, 2011]. This design work is generally geared toward the reuse of well-known topologies in new circuit functions or to improve existing circuit topologies [Jiao and Doboli, 2015] and, less often, to create new circuit topologies. But these procedures not always achieve the best performance within the shortening design-to-market cycle time. Furthermore, as the technology is heading to deep sub-micrometer sizes, the present circuit topologies and design methods are often not good enough to completely fulfill the final circuit high-end specifications. It is imperative to develop a systematic and automatic method to generate new circuit topologies that combines high efficiency with complete specs fulfillment, accommodates submicron technology constraints, and reduces the trial and error cycle of the circuit design life cycle.

Typically the process of analog circuit design for Integrated Circuits (ICs) has three distinctive stages: choice of topology, component sizing and layout generation [Sorkhabi and Zhang, 2017, Lourenço et al., 2016, McConaghy et al., 2009, Žiga Rojec et al., 2019]. For some years now, the synthesis of digital circuits has been almost fully automated [Faragó

et al., 2014], but analog ICs or the analog part of modern Mixed-Signal System-on-a-Chip (MS-SoC) still take a lot of human intervention in the topology choice stage in spite of the research effort of the past few decades in the automated synthesis field. The typical analog circuit design flow [Sorkhabi and Zhang, 2017, Lourenço et al., 2016] includes those three phases with a highly iterative nature, which can be graphically represented in a diagram such as the one shown in Figure 1.1 [Campilho-Gomes et al., 2024].

Typically, in analog circuit design, the topology selection stage is performed by a human designer who, based on experience and knowledge, selects a topology that is believed to provide the desired performance for the given task. The parameters of the circuit components (such as resistance, capacitance, transistor length and width) are then sized so that the circuit achieves the desired performance. This is often an iterative process that involves changing one or more parameters, running a simulation, and evaluating the new behavior of the circuit. If the circuit does not achieve the desired performance, it may be necessary to select a different topology before repeating the sizing step. If this cycle converges to generate a circuit that meets the specifications, we then move on to the layout design stage. When things get complicated, it may be necessary to include the sizing stage in this cycle, and in extreme cases even the topology selection stage may be called into play (although it is more common to make only minor adjustments to the topology than to select an entirely new one).

Since the early 2000s, when the System-on-a-Chip (SoC) "movement" was considered an emerging phenomenon [Linden and Somaya, 2003], up to present days, when the popularity of MS-SoCs is still growing [Datta et al., 2019], the market pressure for the development of SoCs and MS-SoCs keeps increasing, driven by the conjunction of consumer demanding and by the possibilities made feasible by new submicron technologies. The demand for high performance embedded SoCs and MS-SoCs is steering designers to integrate increasingly more IP cores on a single chip [Obaidullah and Khan, 2017] and to explore innovative designs compatible with CMOS technology [Datta et al., 2019]. Many circuit topologies used today have evolved in line with the reduction in CMOS device dimensions and have been optimized according to the currently available technology; however, conventional planar CMOS devices are approaching their scaling limits [Ratnesh et al., 2021].

The Electronic Design Automation (EDA) field, a trend in research effort for the past three decades, targeted in technology roadmaps like International Technology Roadmap for Semiconductors (ITRS) [ITRS, 2019, Kahng and Koushanfar, 2015], focuses in reduction of designer effort, minimization of time-to-market and minimization of production costs [Farago et al., 2015]. Included in EDA, the subfield of Analog Design Automation (ADA) deals specifically with analog circuits, and it also contains the three stages mentioned above: topology synthesis, component sizing and layout generation, though when the circuit is not to be integrated the layout stage is replaced by a Printed Circuit Board (PCB) generation stage.

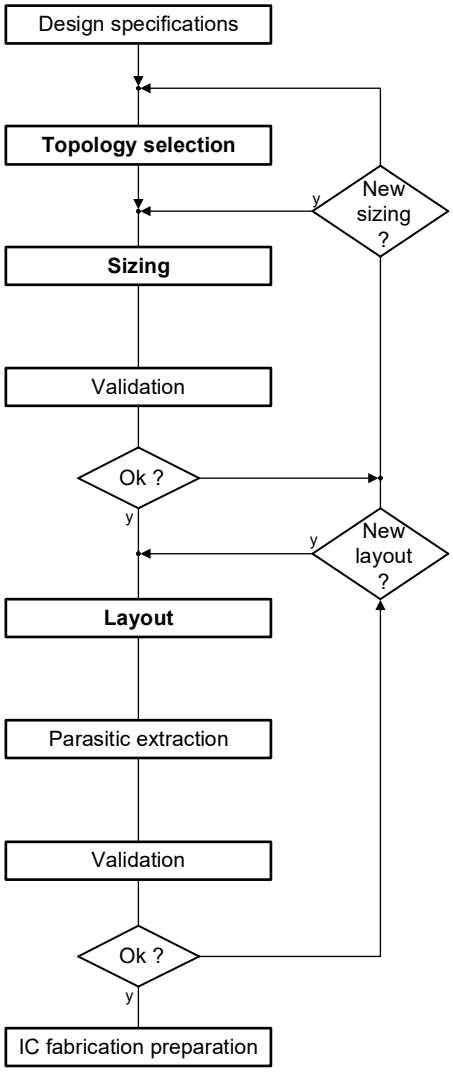


Figure 1.1: Typical analog circuit design flow.

The topology synthesis stage is the least developed stage in ADA. As the authors of [Sorkhabi and Zhang, 2017] point out, "In most of the recent literature, topology selection is viewed as an inferior choice that has stalled since the beginning years due to its severe weakness (e.g., high computational effort and limited beneficiary applications)". Automatic topology synthesis of analog circuits is far from being a straightforward task and, despite having received some research effort in recent decades, especially in the field of Evolutionary Algorithms (EAs), it has not yet matured enough to become available in software packages for the integrated circuit industry.

Previously, a divide-and-conquer strategy was used, as in [El-Turky and Perry, 1989]. In this implementation the circuit is decomposed in predefined basic blocks, *e.g.*, current mirrors; input stage; output stage. In each block, at transistor level, the device sizes are computed. Each basic block topology is selected using artificial intelligence (AI) rules in combination with lookup tables. Setup time is the main disadvantage, since it is necessary to create the AI rules and the lookup tables, or to adjust the existing ones, before design starts. In this approach, the design problem does not consider the complete circuit as a whole, but rather as a computation of sub-block tasks.

Another method of circuit synthesis, proposed in the late 1990s, includes a hierarchy based approach design [Harjani et al., 1989]. This type of approach decomposes the system into system blocks, then, each block into circuit topologies which then are sized. The circuit-level sub-blocks are based on well-known, predefined, topologies that are sized to match the circuit-level specifications computed to the respective sub-block. The computation of sub-blocks specifications at higher level is based on a model of the sub-blocks that does not take into consideration all the higher-order effects and some errors can be introduced. This propagates specification discrepancy from system top level to circuit level. Moreover, each sub-block at circuit level is designed independently, which can lead to a process of back-annotation, if significant discrepancy between the estimated and computed circuit performance results exists. Even considering the reuse of the knowledge of sub-blocks already gathered, it requires a considerable time to design a new circuit.

Generally, and as an example, the state-of-art of amplifier circuit generation approaches rely on performance estimation based on frequency-domain analysis, *e.g.*, open-loop gain (AOL), Gain-bandwidth product (GBW), Output voltage swing (OS), Slew-rate (SR), Power Supply Rejection Ratio (PSRR), Common-mode Rejection Ratio (CMRR) just to mention some. Although some of these parameters may be calculated explicitly, some parameters can not be calculated in an easy way, typically, resulting in an unconstrained problem with too much degree of freedom [Gielen and Rutenbar, 2000].

A time-domain analysis based methodology can significantly simplify the calculus for circuit performance estimation. Considering the example of an amplifier, the time-domain evaluation consists of, besides power dissipation and die area, the settling-time for a given settling accuracy. Moreover, when a given settling-error is reached within a desired settling-

time, it is automatically guaranteed that the amplifier has enough AOL, OS, SR, closed loop bandwidth and closed loop stability.

For example, in switched-capacitor circuits the objective is to have a stable amplifier with a given settling error, after a given available time. By analyzing the step response of the amplifier it is possible to obtain a single key performance indicator (KPI) that encloses all the traditional indicators, such as DC gain, GBW and phase margin (PM) [Santos-Tavares, 2010]. Following this approach, the amplifier design can be accepted just by checking if the settling error is smaller than the desired value and that the closed-loop step response is stable.

In this work research effort is focused on a generic flat circuit-level circuit generator. This circuit generator is expected to handle circuit device elements, *e.g.*, transistors and resistors, and combine them to generate the complete circuit. Generically, the flat circuit-level approach to automatic circuit synthesis is a more challenging task than the reuse and cell-based approach, and becomes even more challenging as the complexity of the circuit increases. But the flat circuit-level approach is probably more prone to generate new topologies and achieve better results, even in the leading technologies.

1.2 Research question, hypothesis and approach

The generation of circuit topologies can be seen as a search and optimization problem and, therefore, it can be performed with Genetic Algorithms (GAs). What is unclear is to what extent this can be accomplished by a GA, particularly when handled at flat circuit-level and in concrete for the case of CMOS amplifiers. If some human designed topologies can be generated by GAs, perhaps these may also generate new topologies. As a natural result of the development of amplifier design conducted to this day and also boosted by the current submicron technologies used in today's ICs, CMOS integrated amplifiers have high efficiencies and very optimized specifications compared to the not-so-distant past. The main research question adopted in this work is therefore:

> Is it possible to generate new circuit topologies for CMOS amplifiers, at flat circuit-level, in an automatic way (software generated), with optimized specifications and high efficiency at submicron technologies?

And the proposed hypothesis to address this problem and to guide the research effort is:

> It is possible to develop a methodology and a software tool based in GAs to, automatically, generate novel flat circuit-level topologies for CMOS amplifiers with high efficiency at submicron technologies.

The research question and the proposed hypothesis enclose some other questions that will be addressed in the course of this work, such as:

- To what extent does the canonical or conventional Genetic Algorithm (GA-c) is suited

to circuit synthesis?

Since Holland's presentation of GA-c there have been many developed variants for the algorithm, and some may be more suitable for circuit synthesis than others. It is unclear what are the actual limitations of GA-c when applied to analog circuit synthesis, particularly to amplifier synthesis, so it is relevant to try to get some sense of what can actually be accomplished by the GA-c before any of its variants are considered.

- What circuit codification is best suited for analog circuit synthesis, namely for amplifier topology synthesis?

  It is well known that part of the difficulty of obtaining good results with GAs is to choose a good codification scheme of the problem. Several coding schemes can be used for electrical circuits, and it should be evaluated which one is best suited for analog circuit synthesis in the specific context of amplifier synthesis.

- How multi-objective algorithms can contribute to better accomplishment of amplifier specifications?

  In amplifier design there are often conflicting objectives which can be handled in more than one way. One such way is by using multi-objective search and optimization algorithms, so it should be evaluated if it is possible to improve the performance of the synthesis of amplifiers with these algorithms.

- How can variable-length chromosomes (VLCs) contribute to the GA's ability to encode circuits, and to what extent does it affect the quality of the topologies produced?

  In GA-c all chromosomes have fixed length (constant quantity of genes) during the execution of the algorithm. This imposes some restrictions on how the circuits are encoded, namely it can restrict the existence of a variable number of circuit nodes and circuit components, and thus may interfere with the GA overall performance in circuit synthesis. A possible alternative is to use the VLC variant of GA, which requires some changes in GA-c, as in the crossover operator, but which can allow an elastic representation of the circuit by providing a mechanism capable of encoding circuits of different sizes.

- How parallelization, and what kind of parallelization, can contribute to the outcome of a GA when applied to circuit synthesis.

  GAs may be highly parallelized and developed software to implement them should use this characteristic. This is an important aspect of this work because the simulation time may be a burden impossible to cope for in the context of a typical PhD time frame. There are several models of parallel programming with threads and a few must be considered, namely the boss/worker model, the peer model and the pipeline model. Because of performance issues, the use of a thread pool design technique

should be considered instead of the more usual thread create-destroy technique which adds too much overhead to the process. Operating systems already optimize several thread-related scheduling issues, but some parameters should be tuned to specific software and hardware characteristics. For instance, the optimum number of simultaneous running threads is strongly hardware dependent, specifically with the number of available CPU cores.

1.3 Aimed contribution

The area of circuit synthesis had significant advances in the last two decades, but there is still a lot of open space for development, with much to be done until one can present tools that are able to replace a human designer in the synthesis of a representative and useful number of analog topologies. Circuit synthesis with GA is still an open field for research, with some published results in analog topologies, but few in amplifiers.

The design of analog circuits is typically accomplished by porting known topologies to new fabrication technologies or circuit functions, or by optimizing existing circuit topologies, and less often by creating new circuit topologies. In fact, due to the high design complexity involved, the huge design space to be explored, the considerable design expertise required, and the trade-offs between conflicting constraints, there are still no widely accepted solutions for automated analog circuit synthesis, even at the research level [Zhao and Zhang, 2020].

According to [Zhao and Zhang, 2020], in terms of implementation strategy, the existing works on circuit topology generation can be classified into three categories: 1) knowledge-based; 2) EA-based; and 3) construction-based; and none of these three categories is absolutely perfect for circuit topology generation. It is difficult to say whether one of these categories is better than the others, and research is still being done in all three categories. The work presented in this thesis focuses on a variant of the so-called "string-based topology generation" [Sorkhabi and Zhang, 2017], which is a problem codification scheme for topology generation that falls into the EA-based category. In string-based topology generation, a circuit is mapped onto a chromosome, which is an array of genes corresponding to circuit components, and a GA is tasked with generating the topology. This method simultaneously sizes components and generates the topology, a process frequently employed in EA-based methods. The results of this process are then used to decide on future changes to the topology.

This work aims to contribute to the pursuit of this line of research, with the intention of finding out whether it would not be possible to go further with GAs in the automatic synthesis of circuits, far beyond passive circuits, in particular for the generation of amplifier circuits, without using any previously known building blocks as a way of promoting novelty in topology generation. In order to provide answers to the research question, developed software that implements a GA will be used to attempt the generation of analog circuits,

namely of amplifiers. The main desired contribution of this work is a software tool capable of generating amplifier circuits, given some specifications. Some sub-topics related to this theme should be addressed which should generate their own contribution, such as:

- Evaluation of the feasibility of generation of topologies at a flat circuit-level, *i.e.*, at the component level like transistor, resistor and capacitor, with GAs.
- Evaluation of the adequacy of time domain analysis for amplifier synthesis.
- Evaluation of VLC adequacy in the context of analog circuit synthesis.
- Propose a suitable circuit codification scheme for GAs to be used in amplifier synthesis.
- Evaluation of the adequacy of adopted model for parallelization of the GA.
- Evaluation of the need of multi-objective variants of GAs for amplifier synthesis.
- Evaluation of the ability of the software tool to generate new amplifier topologies.

Other expected contribution is added insight on various topics related to practical details of the implementation of GAs in the context of analog circuit synthesis. Also, added insight into limitations of GAs used in this context will inevitably arise and will be exposed in the course of this work.

CHAPTER 2

Literature Review

This chapter presents a brief literature review in several topics related to this work, namely genetic algorithms and its use in circuit optimization and synthesis.

2.1 Electronic design automation

Electronic Design Automation (EDA) field has been been a key enabler of the semiconductor industry's growth for the past 50+ years, that enabled the modern computing era [Kahng and Koushanfar, 2015]. It is a truly interdisciplinary field: "its abstractions, computational models, algorithms, methodologies and tools" have drawn knowledge from a number of areas like electric engineering, computer science, applied mathematics, operational research, chemistry and physics [Kahng and Koushanfar, 2015], just to name the most prominent. EDA field comprises all software tools that work together in the design flow of an IC, performing synthesis, optimization, simulation and verification tasks across all of the intervening levels of abstraction [Kahng and Koushanfar, 2015].

Inside the broad field of EDA is the subfield of ADA and in it there is one narrow area of automatic analog circuit synthesis which aims the generation of circuit topologies by software tools that received the circuit specifications. As mentioned in Chapter 1, in the context of the integrated circuit industry, ADA is usually described as containing three stages: topology synthesis, component sizing and layout generation. The two later stages are more developed than the former one, which still requires a lot of human effort.

Analog circuit design is often considered to be an art [Williams, 1998, Ferent and Doboli, 2011], made possible by solid engineering skills, designer talent, large knowledge of building blocks and great ease in design reuse, all acquired through many years of experience [Gielen and Rutenbar, 2000]. Therefore, and not surprisingly, analog circuit design automation is frequently referred to as a non-trivial task, and when compared to digital circuit automation it is considered a harder task. Over the years, several authors often refer these characteristics, either in a more objective and technical way [Harjani et al., 1989, Carley et al., 1996, Gielen and Rutenbar, 2000, Jiao et al., 2015, Maji et al., 2016], or in a less formal, even empathetic, way [Ferent and Doboli, 2011].

2.1.1 Analog vs digital design automation

In [Gielen and Rutenbar, 2000], Gielen and Rutembar say that "[...]analog design in general is perceived as less systematic and more heuristic and knowledge-intensive in nature than digital design, [...]" and that "Analog IC design is a complex endeavor, requiring specialized knowledge and circuit design skills acquired through many years of experience." Continuing the comparison with digital circuits, they claim that "[...]analog circuits are more sensitive to nonidealities and all kinds of higher order effects and parasitic disturbances[...]" and conclude that "These differences from digital design also explain why analog CAD tools cannot simply adapt the digital algorithms, but why specific analog solutions need to be developed that are target to the analog design paradigm and complexity."

Earlier, back in the eighties, Harjani had already pointed out that "[...] it is clear that we cannot directly import design methodologies from the digital domain into the analog domain without significant changes." [Harjani et al., 1989]. He also mentioned that in digital design it is easier to isolate the higher levels of design from device level parameters and layout constraints, and that in analog design it is almost the opposite, since the higher levels of design can be quite markedly affected by nuances in the IC process.

2.1.2 Analog or digital? Both

Despite the enormous growth that the digital domain has had in recent decades in the IC industry, highly driven by the microprocessor and memory market, the Analog and Mixed-Signal (A&MS) ICs domain also grows, largely driven by the mobile phone market for the last two decades and, more recently, much driven by the Internet of Things (IoT), energy management, automotive, and medical markets [Martens and Gielen, 2008, Corp, 2015] [Martins et al., 2017, pp. 4–6] [McGrath, 2018].

As some authors have pointed out over the years [Gielen and Rutenbar, 2000, Lohn et al., 2000], "[...]there are some typical functions that will *always* remain analog." [Gielen and Rutenbar, 2000] and it is clear that "[...]analog circuits are indispensable in all electronic applications that interface with the outside world[...]" [Gielen and Rutenbar, 2000], because "[...] world is fundamentally analog in nature." [Lohn et al., 2000] and

although it is probably true that "[...] the amount of digital design activity far outpaces that of analog design,[...]" [Lohn et al., 2000] it is also true that "[...] most digital systems require analog modules for interfacing to the external world." [Lohn et al., 2000].

2.1.3 Analog and digital: CAD tools

In their year 2000 survey about Computer-Aided-Design (CAD) tools for analog and mixed-signal integrated circuits, Gielen and Rutembar [Gielen and Rutenbar, 2000] state that "In the digital domain, CAD tools are fairly well developed and commercially available[...]" and that, by then, "[...] many lower-level aspects of the digital design process are fully automated". In contrast, "[...] the story is quite different on the analog side. There are not yet any robust commercial CAD tools to support or automate analog circuit design apart from circuit simulators [...] and layout editing environments and their accompanying tools [...]."

Techniques for analog circuit design automation began appearing in the eighties [Lohn and Colombano, 1998] and there are CAD tools for A&MS circuits commercially available for (at least) the past two decades [Harjani et al., 1989,Carley et al., 1996,Ferent and Doboli, 2011] but most deal with simulation and optimization routine tasks, like component sizing and layout constructing (editing and verification), and some offer component sizing while being layout aware, which means that the optimization includes parasitic effects of the integration process [Doboli and Vemuri, 2003,Martins et al., 2012,Rocha et al., 2014][Martins et al., 2017, pp. 4–6]. Until late eighties, the design of the analog section of a mixed-signal Application Specific Integrated Circuit (ASIC) was made manually by the so called experts, while the digital section was already designed, mostly automatically, by CAD tools [Harjani et al., 1989]. But few CAD tools offer some kind of circuit synthesis for A&MS circuits.

2.1.4 Analog circuit synthesis

Works in analog circuit synthesis have been published since the eighties up to the present. In 1989, R. Harjani developed a framework for analog circuit synthesis [Harjani et al., 1989] in which CMOS operational amplifiers (opamps) and comparators were synthesized. This framework is a knowledge-based analog circuit synthesis tool in which circuit topologies are represented as a hierarchy of abstract functional blocks each with associated design knowledge. In fact, this system relies on the existence of a number of mature design topologies (called "styles" in this work's nomenclature) at each level of the hierarchy, which are an interconnection of abstract sub-blocks.

For instance, if a single-stage opamp was to be built, an existing circuit topology would be selected, eventually consisting of a differential pair and several current mirrors as shown in Figure 2.1. The system would then search in the knowledge-base for the sub-blocks (differential pairs and current mirrors) that best fit the required performance and then would try to size their components. The selection and sizing phases are integrated in an iterative process.

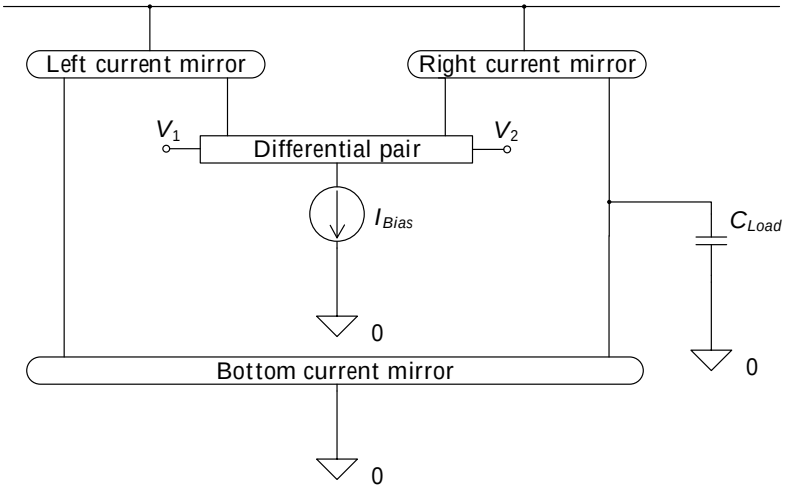


Figure 2.1: Harjani OTA circuit topology (adapted from [Harjani et al., 1989])

Another knowledge-based approach was presented in the same year in [El-Turky and Perry, 1989] which also relies on well known and matured circuit topologies to build the final circuit, differing from Harjani's framework in the hierarchical abstraction and sub-blocks selection process, which is based on heuristics on both works.

In [Kruiskamp, 1995], topology selection and circuit sizing are performed simultaneously by means of a GA. Like the previous works already mentioned, this is also a knowledge-based approach since topologies are made up from basic building blocks. They used the proposed methodology to synthesize opamps, assuming that the topology of an opamp can be separated into three building blocks (an input stage, an optional second gain stage and an optional output buffer) and providing several different topologies for each building block.

Three years before, in 1992, Koza and his colleagues [Koza et al., 1992] introduced Genetic Programming (GP), which they described as an extension of the GA developed by Holland [Holland, 1975] (in which the population consists of computer programs) and in 1997 they applied GP to automate the synthesis of analog circuits [Koza et al., 1997a]. In their work several analog circuits were successfully synthesized without resorting to a knowledge-base. Each circuit had an embryonic circuit that was evolved by the algorithm until, eventually, its pretended specifications were fulfilled. The SPICE simulator was used to evaluate the circuits throughout the evolution process, either using AC small signal analysis, operating point analysis or DC sweep analysis, as convenient.

One of the synthesized circuits by Koza and his colleagues in [Koza et al., 1997a] was a crossover (woofer and tweeter) filter having a single input and two outputs, which had the embryonic circuit depicted in Figure 2.2. In this circuit, only connections between points A, B and C (*i.e.*, connections $\widehat{AB}$, $\widehat{AC}$ and $\widehat{BC}$) can be modified by the evolving algorithm.

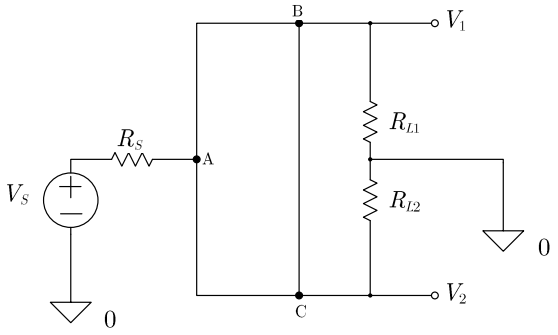


Figure 2.2: Crossover embryonic circuit (adapted from [Koza et al., 1997a]).

That embryonic circuit was evolved until the pretended specifications were fulfilled and the resultant circuit is shown in Figure 2.3.

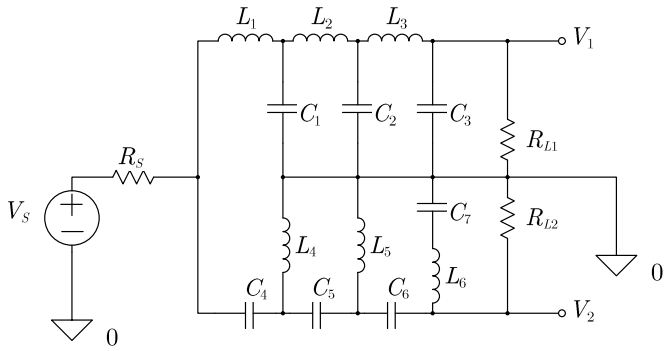


Figure 2.3: Crossover evolved circuit (adapted from [Koza et al., 1997a]).

One other synthesized circuit in [Koza et al., 1997a] was a circuit that produces an output voltage equal to the cube root of its input. This circuit, composed of transistors, diodes, resistors, and capacitors, is called a *computational circuit*. In the same year of 1997, Koza and other researchers published another article entirely dedicated to the synthesis of computational circuits [Koza et al., 1997b]. In this work they applied GP for both the generation of the circuit topology and the sizing of all circuit components, and successfully synthesized circuits for the cube root, square root, square, and cube functions, using a fitness measure based on SPICE's DC sweep applied to the evolved circuits.

On another article published in the same year [Koza et al., 1997c], Koza and others also published the results of one work with which they synthesized an amplifier using bipolar junction transistors (BJTs), resistors and capacitors. Again, they used SPICE's DC sweep to evaluate all evolved circuits.

These works of Koza were a major breakthrough in the automated synthesis of analog

circuits. Using GP, several circuits were successfully synthesized without resorting to a knowledge base, having in their origin only a simple circuit called *embryo circuit* and using SPICE to evaluate the synthesized circuits.

In Koza works, the evolutionary algorithm (*i.e.*, GP) has the same *Darwinian* base as Holland's GA but differs substantially from Holland's algorithm in the way individuals (circuits) are codified, because in GP individuals are codified as programs, with a methodology called "Circuit Constructing Tree". Holland's GA, also referred in the literature as GA-c, encodes individuals ('chromosomes', in Holland's terminology) as strings of '1s' and '0s'.

GP is nowadays applied to many fields of engineering, besides being used in the synthesis of electronic circuits. In this field of circuit synthesis, the works of Jonh Koza are probably the most well known and cited (particularly those published before year 2000 [Koza et al., 1997a, Koza et al., 1997b, Koza et al., 1997c] but also others published later [Koza et al., 2000, Mydlowec and Koza, 2000, Koza et al., 2008]), however there are other authors with published work that also use GP in the synthesis of electronic circuits [Sripramong and Toumazou, 2002].

In [Zebulum et al., 1998], R. Zebulum and others were able to synthesize two analog filters, a low-pass filter and a stop-band filter, using a GA. They used fixed-length chromosomes (FLCs), as is specified in Holland's GA, but also tested for VLCs, which means the codification of individuals does not have a constant number of genes. For circuit evaluation they used SPICE and, in some cases, "[...]a hand written simulator, the SMASH simulator, from Dolphin corporation[...]" [Zebulum et al., 1998].

In the same year, J. Lohn and S. Colombano [Lohn and Colombano, 1998] also used a GA to synthesize analog filters (two low-pass filters), employing VLCs. They tested a technique with a different codification paradigm than Zebulum, in which they codified component values and placement altogether, and also created their own mutation operators, adapted to the codification procedure. As Zebulum, they used SPICE to evaluate evolved circuits. In this proposed technique, the circuit is constructed between fixed input and output terminals, as shown in Figure 2.4, which in fact constitutes an embryonic circuit (depicted components in Figure 2.4 remain unaffected by the evolution process, which affects only the connection between points A and B). The resultant circuit after evolution is shown in Figure 2.5.

Although their technique is topology-limited (as they clearly state), the ability of the proposed system to produce useful circuits was demonstrated.

These works [Zebulum et al., 1998, Lohn and Colombano, 1998], both published in 1998, one year after the works of Koza and colleagues, are, as well as Koza's works, important milestones in the application of evolutionary strategies to the synthesis of analog circuits, not only because the GA was applied successfully to the synthesis of analog circuits but also because the VLC technique was introduced in circuit synthesis, also successfully.

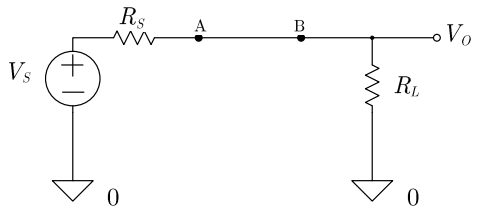


Figure 2.4: Low-pass filter embryonic circuit (adapted from [Lohn and Colombano, 1998]).

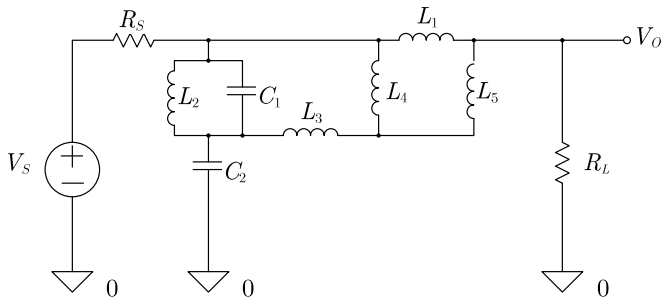


Figure 2.5: Low-pass filter evolved circuit (adapted from [Lohn and Colombano, 1998]).

Many subsequent papers published by numerous researchers are based on those late 1990s works of Lonh and Koza (and colleagues). Both approaches, either using GAs or GP for synthesis of analog circuits were pursued and, eventually, improved in some way.

In 2001, C. Goh and Y. Li [Goh and Li, 2001] pursued the synthesis of analog filters based in GA. They used FLCs, employing a special gene they called "NULL" to encode the absence of a component, which allows a variable number of components in the real circuit. They also introduced new operations to be executed in the mutation phase of the algorithm.

In the same work the authors also tested embryo circuits embedded with *a priori* knowledge of the pretended outcome. They considered it a better starting point for the evolution of circuits because it reduced the number of faulty circuits produced during the evolutionary algorithm execution, thus reducing the time taken for the whole process to find a suitable circuit. However, as other authors point out [Lohn et al., 2000, Yuan and He, 2010], assuming little or no prior knowledge is not necessarily an algorithm weakness. It may be an advantage for evolutionary design algorithms since it may enhance the exploration of novel designs leading to new circuit topologies.

Later, in 2006, Y. Sapargaliyev and T. Kalganova [Sapargaliyev and Kalganova, 2006a, Sapargaliyev and Kalganova, 2006b,Sapargaliyev and Kalganova, 2006c] tried to reduce the computational effort allocated to the so-called preprocessing phase (therefore reducing the total execution time of the synthesis procedure). This preprocessing phase is a procedure

that precedes the evaluation of a circuit and is responsible for validating circuits for simulation (it tries to avoid initiating the simulation of circuits that the simulator considers invalid). This preprocessing procedure is usually based on verification of several circuit-structure-checking rules, avoiding submitting invalid circuit netlists to simulation.

When an invalid circuit is spotted in that preprocessing phase, several procedures may follow. For example, some authors choose to "throw away" the circuit, evaluating it with a very penalizing evaluation fitness value, but other authors try to fix the problem so that the circuit becomes fit to be submitted to simulation. However, in some works of Sapargaliyev and Kalganova [Sapargaliyev and Kalganova, 2006a, Sapargaliyev and Kalganova, 2006b, Sapargaliyev and Kalganova, 2006c] the preprocessing phase was reduced to a minimum, and their version of a GA still managed to successfully synthesize several analog filters.

The authors have called *absolutely unconstrained evolution* to this circuit netlist generation process during which no circuit-structure-checking rules are applied and all circuits are submitted to simulation. However, as the authors point out, they still applied two circuit-structure-checking rules: detection of elements with dangling nodes and with isolated subcircuits. And for these circuits the authors still used some "mending" procedures, connecting dangling nodes to ground with 1 GΩ resistors and removing isolated subcircuits.

Combined with mutation operators defined by the authors, the VLC technique was used in these works, either in a mode in which the size of the chromosome can increase but cannot decrease, which the authors called *increasing length genotype*, or in a mode in which the size can either increase or decrease, that the authors called *oscillating length genotype*.

In 2010, the same authors tried the synthesis of other analog circuits in addition to filters, also based in GAs. In [Sapargaliyev and Kalganova, 2010a] they succeed in synthesizing an analog circuit with 138 components. These components were chosen among bipolar transistors (either npn or pnp), resistors, capacitors and inductors and both the topology and the component sizing were evolved by the GA. This circuit, an 8-output voltage distributor, which had its embryonic circuit as shown in Figure 2.6, was synthesized by a GA 'tweaked' with a few 'novelties' if compared to the GA-c (*i.e.*, Holland's original, or canonical, GA). The *oscillating length genotype* was used and the mutation phase included a substructure reuse operator. This operator copies part of a circuit to another location of the actual circuit and is able to replicate it entirely, which is very handy in this problem since it enables a fast creation of a subcircuit that generates another output.

In fact, using this feature, the 8-output voltage distributor circuit synthesized in [Sapargaliyev and Kalganova, 2010a] was evolved in eight steps: first the circuit that generates one output was evolved, and when this task was completed the whole circuit was cloned, so that another output was generated. This circuit would then undergo another evolution stage in order to adjust itself to its specifications. This process was repeated until all eight outputs were generated and the authors called it *incremental evolution*. During the overall synthesis of all eight circuits, the fitness function is being adjusted in order to include the

specifications of each output, so the authors called this fitness function a *dynamic fitness function*.

The circuit to be synthesized in [Sapargaliyev and Kalganova, 2010a] is an active circuit, so the respective embryo circuit would need some sort of power supply. This one had two DC voltage sources available, one of 1.5 V and another of 15 V (as depicted in Figure 2.6), and it was up to the GA to "choose" either one or both sources.

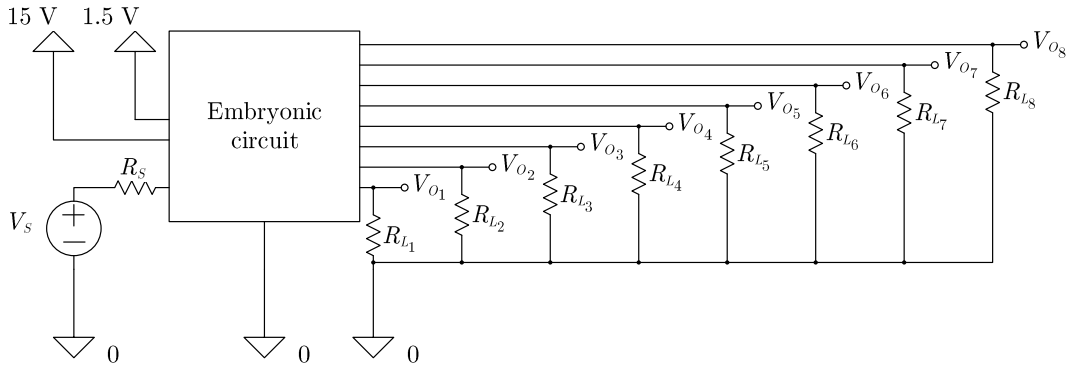


Figure 2.6: 8-output voltage distributor embryonic circuit (adapted from [Sapargaliyev and Kalganova, 2010a]).

Using the same techniques, like the substructure reuse already referred, applying *oscillating length genotype*, and "tweaking" the GA with novel mutation operators, the same authors published (in the same year of 2010) interesting results regarding the synthesis of some analog computational circuits [Sapargaliyev and Kalganova, 2010b]. Like Koza did in [Koza et al., 1997b], they also synthesized circuits for the cube root, square root, square, and cube functions, but instead of using GP (like Koza did) they used a GA.

In [Sapargaliyev and Kalganova, 2010b], all four computational functions were successfully accomplished by evolved circuits using only resistors and bipolar transistors (either npn or pnp) and they all performed better, the authors claim, when compared to Koza's evolved circuits, namely because they used less components and had less averaging error. As well as the embryonic circuit of Figure 2.6, the embryonic circuit used for the synthesis of these computational circuits also had power supply sources available, although in this case the authors used two symmetrical DC voltage sources of 15 V and −15 V, as shown in Figure 2.7. Circuit fitness evaluation was done by PSPICE performing transient analysis using an input voltage source swept linearly in 0.2 s from 0 to 500 mV for the square root function and from −250 mV to 250 mV for the other functions.

As an example, the evolved circuit for the square root function is depicted in Figure 2.8. This circuit shares its 'oddity' with many evolved circuits. Evolved circuits often have unconventional designs and are not always easy to understand [Thompson and Layzell, 1999, Sapargaliyev and Kalganova, 2010b] and they seldom "[...]produce novel yet useful topological design features" [Ferent and Doboli, 2011]. Nevertheless this circuit fulfills

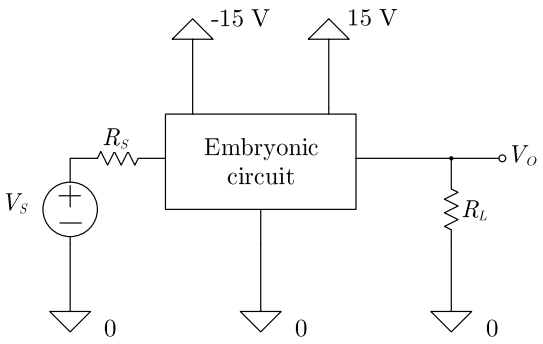


Figure 2.7: Square root embryonic circuit (adapted from [Sapargaliyev and Kalganova, 2010b]).

the intended specifications and is evidence that the proposed methodology can produce functional circuits.

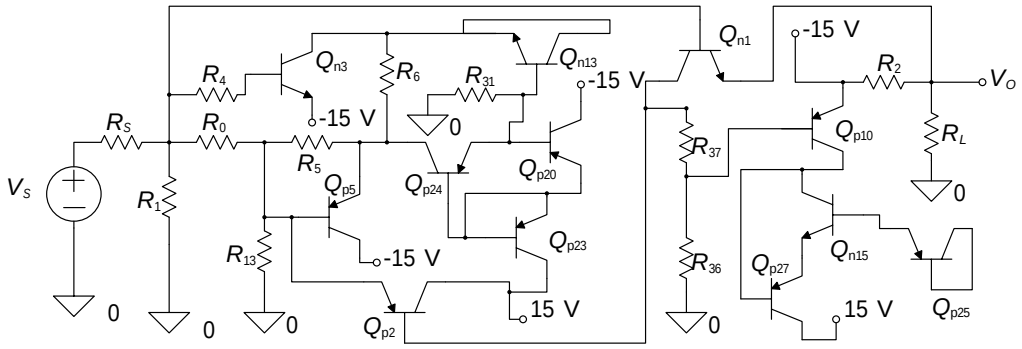


Figure 2.8: Square root evolved circuit (adapted from [Sapargaliyev and Kalganova, 2010b]).

Other computational circuits were successfully synthesized by H. Karci and colleagues in [Karci et al., 2016]. Using a modified GA they tested the synthesis of circuits for the Gaussian, sigmoid, cube and square functions. The aim of the paper was to introduce a modification to the GA-c that would improve its local search capability. The authors called this a 'speciation' procedure, which is in fact a new selection operator that the authors claim to produce better results without additional circuit simulation, which is usually the large computational load of a GA oriented for circuit synthesis. Regardless of the appreciation that can be made about the real improvement achieved with this new methodology, this work renews and reinforces the evidence of the potentiality of GAs in circuit synthesis.

Although GAs have been and continue to be used by many researchers in analog circuit design automation, and while many published works focus on a single algorithm, there are also works that use multi-algorithm approaches [Liu and He, 2012] and others where the

authors looked for other methodologies to tackle the problem.

2.1.4.1 Other trends in circuit synthesis

Although Holland's work published in 1975 is undoubtedly a very important milestone in evolutionary computation methods, earlier works have been proposed that employ Darwin's evolution theory [Box, 1957, Bremermann, 1962]. It is mainly due to recent advancements in computer technology that much research effort has been channeled into the field of GAs, with emphasis in the development of new GA-based optimization strategies [Zhang et al., 2007].

In the last two decades, CAD tools for circuit sizing and layout design had a significant advance but devising or refining circuit topologies remains difficult [Jiao et al., 2015]. GAs and other evolutionary algorithms [Maji et al., 2016] played a relevant role in this advance and although these approaches "[...] can, in theory, evolve any circuit topology, in reality, creating performance-efficient yet minimal structures is hard as it involves searching an open-ended, widely extensible, and strongly discontinuous solution space." [Jiao et al., 2015].

So, although the evolutionary algorithms path is still being explored for circuit synthesis [Karci et al., 2016, Ushie et al., 2017], other methodologies are also being presented and tested by some authors. In [Ferent and Doboli, 2014a], Ferent and Doboli propose a new methodology to describe a circuit (named Ordered Node Clustering Representation (ONCR)). This methodology is in fact a graph that allows for identification of structural similarities and differences of circuits.

In the same year, in [Ferent and Doboli, 2014b], a new approach for circuit topology selection and refinement is proposed by the same authors, which is based in a reasoning-like process using the ONCR methodology. In this approach, circuit features are selected in successive steps from a large pool of existing circuit designs in order to improve the circuit performance. The authors claim this procedure emulates designer reasoning by identifying the structure causing performance limitations and attempting to locally modify that circuit topology.

In a not too far away research path, and still in the same year, Doboli and Umbarkar [Doboli and Umbarkar, 2014], through an experimental study grounded in cognitive psychology, try to envisage what is design creativity in electronic embedded systems and what drives it in human based circuit synthesis. They specifically study the influence of what they called "precedents" on the novelty, variety, quality, and utility of design solutions, and define "precedents" as solutions and solution features that are available to solve a problem (including those that become available during the iterative solving process in the context of team work).

One year later, Jiao, Montano and Doboli [Jiao et al., 2015], based in those previous works [Ferent and Doboli, 2014a, Ferent and Doboli, 2014b], propose a new topology

synthesis method. This method does not use a GA and needs a large pool of existing circuit designs, but is capable of generating "circuits beyond the capabilities of existing topology synthesis algorithms", the authors claim, and the synthesized topologies are similar to designer-created circuits.

2.2 Genetic Algorithms in EDA

Genetic Algorithms (GAs) are search and optimization algorithms based on the mechanics of natural selection and natural genetics [Goldberg, 1989, pp. 1]. They were developed by J. Holland and colleagues [Holland, 1975] in the early seventies and today have broad application in the areas of business, engineering and science. A GA is a heuristic method that mimics the natural evolution process by applying what is referred as *Darwinian natural selection* or *the principle of survival of the fittest*, and it has the ability to operate in large, complex and highly constrained search spaces while exhibiting a good capacity to escape local minimum and insensitivity to initial conditions [Farago et al., 2015, Rajasekaran and Pai, 2017].

GAs belong to the larger group of EAs, which constitute a class of stochastic search and optimization methods that simulate the process of natural evolution [Nicosia et al., 2008]. The EAs group is itself part of broader area of computer science concerned with designing intelligent computer systems that is Artificial Intelligence [Rajasekaran and Pai, 2017]. Perhaps one of the most relevant differences between EAs and classical optimization algorithms is that EAs use a population of candidate solutions in each generation, instead of a single solution, and the outcome is a population of possible solutions [Nicosia et al., 2008]. Another difference is that in EAs there is no restriction on the objective function (or functions): it can be non-differentiable or even discontinuous and there is no need to know its exact form, therefore its evaluation can be accomplished by simulation [Nicosia et al., 2008]. For some authors, GAs are generically superior to some classical optimization algorithms, like gradient descent techniques, for locating the global optimal solution [Zhang et al., 2007].

Currently, EAs are unavoidable in EDA tools [Afacan et al., 2021] and are used in circuit synthesis, sizing, and optimization for numerous problems [Passos et al., 2018, Passos et al., 2019, Beaulieu et al., 2023, Afacan and Dundar, 2019, Domingues et al., 2022]. The simulator-in-the-loop paradigm is often used in EDA tools, and is acknowledged the way to go in recent (and also in not so recent) works [Afacan and Dundar, 2019, Domingues et al., 2022, Krasnicki et al., 1999, Phelps et al., 2000].

In EDA, GAs have been used in several of its stages. In [Farago et al., 2015] they were used to optimize two design stages, sometimes referred as "Nominal design" and "Design for yield". The so-called "Nominal design" stage is concerned with "performance", *i.e.*, fulfillment of the circuit specifications, which basically consists in finding a set of component's values of a given circuit topology that satisfy those specifications. "Design for yield" is in fact

design for robustness, commonly related to manufacturing non-idealities, like process imprecisions and silicon crystal defects [Nicosia et al., 2008]. Whereas "Nominal design" stage is often supported by DC, TRAN and AC analysis in SPICE like circuit simulators, "Design for yield" stage is usually supported by Monte Carlo, corner and worst case simulations [Farago et al., 2015].

2.2.1 Genetic Algorithms in circuit synthesis

In a GA there is this notion of *a population*. In GA nomenclature, a population is a set of individuals, also called chromosomes, each one encoding a candidate solution to the problem being addressed. Each chromosome has several variables, also called genes, that build the search space.

Figure 2.9 contains a diagram of a typical application of a GA in circuit synthesis, where the various stages of the algorithm are depicted.

2.2.1.1 Encoding an individual

Starting the algorithm, a population is built by initializing all genes on each chromosome, usually in a randomly fashion. This is the *initial generation*. Typically, in circuit synthesis context, each chromosome encodes one circuit which is a possible candidate to fulfill a set of specifications. Problem representation, which translates to individual encoding in GAs, is a major component [Whitley, 1994] of GAs application to a concrete problem, and therefore encoding a circuit in a GA is a topic frequently discussed and object of significant research effort [Kruiskamp, 1995, Koza et al., 1997a, Lohn and Colombano, 1998, Lohn et al., 2000, Sapargaliyev and Kalganova, 2010b].

2.2.1.2 Fitness

The level of specifications fulfillment performed by each chromosome is assessed at the next stage of the algorithm, entitled *Fitness evaluation* in Figure 2.9, in which a fitness measurement (often a single floating point value) is assigned to each chromosome according to a defined fitness function. This fitness value may be seen as a figure of merit (in which case higher is better, and the value is often in the interval $[0,1]$) or a cost value (in which case lower is better and the value is often in the interval $[0,+\infty[$).

In circuit synthesis context, as well as in several CAD tools for A&MS, fitness computation is frequently accomplished in a software basis, either using a circuit simulator like SPICE (or one of its manifold variants like HSPICE®[Synopsys, 2017], ELDO®[Mentor Graphics, 2009] or NGSPICE[SourceForge, 2019]) or using a somewhat simplified equation set that models circuit components well enough (considering the circuit specifications)[Carley et al., 1996, Vaz et al., 2001, Paulino et al., 2001, Jr et al., 2014, Ushie et al., 2017], or even a mix of these two approaches [Faragó et al., 2014].

In A&MS, hardware based fitness computation is seldom mentioned (in [Torresen, 2004], the author refers the possibility of Field Programmable Analog Array (FPAA) usage but

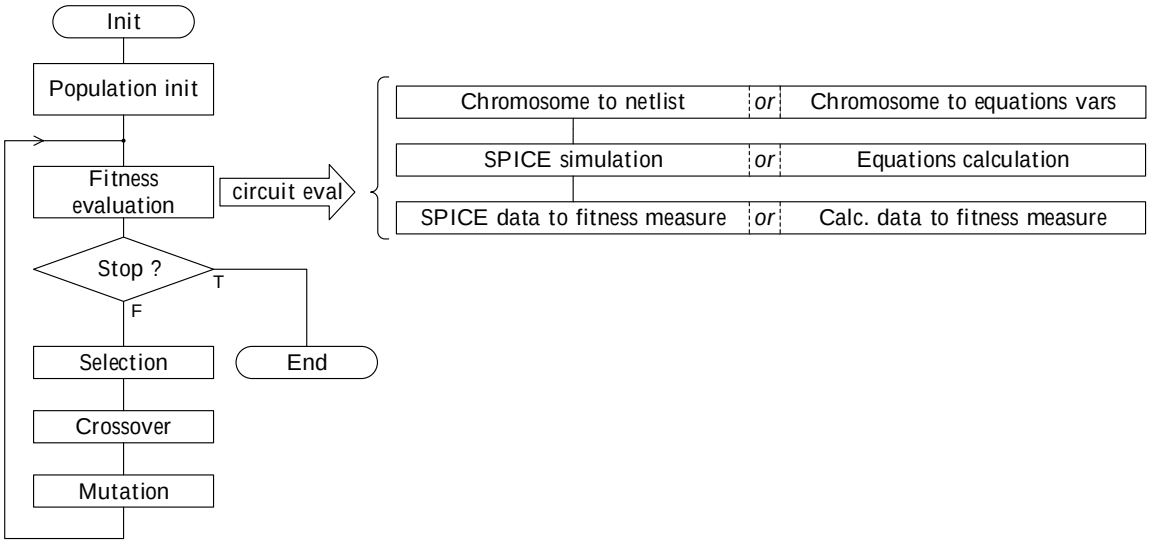


Figure 2.9: Genetic algorithm in circuit synthesis and optimization.

doesn't mention any works that actually uses them). However, in digital circuits, fitness computation is sometimes accomplished directly in hardware, using an Field Programmable Logic Array (FPGA) [Salvador, 2016] or even an ASIC [Torresen, 2004] or, as in earlier works like these [Iwata et al., 1996, Kajitani et al., 1996], using a Field Programmable Device (PLD). This is sometimes called *intrinsic evolvable hardware* or *online hardware evolution* or even *online fitness computation*, because evaluation takes place in the target device, as opposed to evaluation on a computing system other than the final device, which is sometimes called *extrinsic evolvable hardware*, *offline hardware evolution* or *offline fitness computation* [Iwata et al., 1996, Torresen, 2004, Salvador, 2016].

In software based fitness computation systems there are many processes for calculating the fitness of a chromosome that codifies a circuit. Of the most used when there is only one variable to be optimized (*i.e.*, single objective design problems with one decision variable), there are two that stand out by the frequency with which they are mentioned in the literature: one process, represented by function f_1 in eq. (2.1), uses the sum of the absolute value of the difference of the individual's achieved value x_i and the desired output $\widehat{x_i}$, over all N evaluation (or sampled) points [Lohn and Colombano, 1998, Lohn et al., 2000], and the other, represented by function f_1 in eq. (2.2), uses the mean square error [Liu and He, 2012].

$$f_1(x) = \frac{1}{N}\sum_{i=1}^{N}|x_i - \widehat{x_i}| \qquad (2.1) \qquad\qquad f_1(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \widehat{x_i})^2 \qquad (2.2)$$

These objective functions are meant to be minimized by the GA (so they may be implemented without the $\frac{1}{N}$ normalizing factor) and are used in numerous design problems. For instance, the filter presented in Figure 2.5 was synthesized using an objective function

like f_1 in eq. (2.1), using amplitude voltage values from an AC simulation performed in SPICE [Lohn and Colombano, 1998]. In [Zebulum et al., 1998] a similar function was used to synthesize another low-pass filter but a weighted sum was used, in order to emphasize the importance of some frequencies to the detriment of others, so eq. (2.3) was used instead of eq. (2.1).

$$f_1(x) = \frac{1}{N} \sum_{i=1}^{N} W_i |x_i - \widehat{x_i}| \tag{2.3}$$

The weights used in eq. (2.3) are usually constant during the evolution of the circuit but can be adapted (changed) along the execution of the GA [Dendouga and Oussalah, 2015, Serra, 2017]. This can be done to try to guide evolution to first do a "course" search and later a "finer" search, in order to first find a non-optimal solution and later an optimal (or near-optimal) solution. The weight's dependence with the evolution of the circuit, *i.e.*, with the progress of the algorithm, is represented in eq. (2.4) by the weight's dependence over time, $W_i(t)$. In this context, "time" means algorithm *generation* or *iteration*, therefore it has a discrete nature. In the bibliography on GAs, the terms *generation* and *iteration* are often used interchangeably and will also be used as such throughout this document.

$$f_1(x,t) = \frac{1}{N} \sum_{i=1}^{N} W_i(t) |x_i - \widehat{x_i}| \tag{2.4}$$

Another way to try to guide evolution in the GA is to slowly change the desired value towards a more demanding but more relevant value during the GA execution [Serra, 2017]. This can be represented by a time dependence of the desired value, as depicted in eq. (2.5).

$$f_1(x,t) = \frac{1}{N} \sum_{i=1}^{N} W_i(t) |x_i - \widehat{x_i}(t)| \tag{2.5}$$

To try to direct the evolution of a GA with the methodologies afore mentioned is not always a successful endeavor, and must be applied with caution since it can transform a convergent algorithm into an erratic, non-convergent one.

Design problems can have more than one objective function, which may be combined somehow in a single objective function as was accomplished in the synthesis of the two-output filter of Figure 2.3. In this problem the goal was to minimize two functions (each related to each of the circuit outputs) with the format of eq. (2.6) and (2.7).

$$f_1(x_1,t) = \sum_{i=1}^{N} W_{1_i}(t) \left| x_{1_i} - \widehat{x}_{1_i} \right| \tag{2.6}$$

$$f_2(x_2,t) = \sum_{i=1}^{N} W_{2_i}(t) \left| x_{2_i} - \widehat{x}_{2_i} \right| \tag{2.7}$$

Since the algorithm is not prepared to use more than one fitness function, those two functions were aggregated in one, by summing them with equal weights (eq. (2.8)), creating

f_3 which was the actual fitness function that was minimized in [Koza et al., 1997a].

$$f_3(x_1, x_2, t) = f_1(x_1, t) + f_2(x_2, t) \tag{2.8}$$

In eq. (2.8), both f_1 and f_2 contribute in the same way to f_3 but in other problems functions f_1 and f_2 could have different relative "importance" so there could be interest in using a weighted function sum, in which case eq. (2.9) would be used instead of eq. (2.8).

$$f_3(x_1, x_2, t) = W_{f_1} f_1(x_1, t) + W_{f_2} f_2(x_2, t) \tag{2.9}$$

Equation (2.9) can be rewritten in a more generic format (and losing dependence with t for the sake of simplicity) as eq. (2.10), where M objective functions are aggregated into a single one, as is often done to solve many problems with GAs.

$$f_{fit}(\overline{x}) = \sum_{i=1}^{M} W_{f_i} f_i(\overline{x}) \tag{2.10}$$

In GAs, it is common practice to combine objective functions in an overall objective function, often obtained by weighting several goals (but there are other techniques used for the same goal [Bechikh et al., 2017]). This has the effect of concealing the eventual existence of conflicting goals, taking away from designers the freedom to choose among different, equally feasible solutions [Nicosia et al., 2008]. For instance, in IC design, typical objective and constraint functions are gain, bandwidth, area, noise, speed, linearity or power consumption expressed as a function of design parameters, such as transistor sizes, resistor or capacitor values, inductor geometry, etc, and often some of these constraints conflict, *i.e.*, improving one worsens another [Goh and Li, 2002, Nicosia et al., 2008]. So, as any designer knows, trade offs have to be done. Although significant portion of research and application in the field of optimization considers a single objective, most real-world problems involve more than one objective [Deb, 2001, Bechikh et al., 2017] and some of these objectives can interfere with each other with conflicting behaviors.

As mentioned before, one of the most relevant differences between EAs and classical optimization algorithms is that EAs use a population of candidate solutions in each iteration, instead of a single solution, and the outcome is a population of possible solutions [Nicosia et al., 2008]. So, when an optimization problem involves more than one objective functions, there may be several optimum solutions, which should be identifiable in the outcome population. This is not possible in a canonical GA but it is achievable with other algorithms that perform what is known as *multi-objective optimization*.

Inside the group of EAs, *multi-objective optimization* is the subject of Multi-Objective Optimization Evolutionary Algorithms (MOOEAs) (also referred in the literature as Multi-Objective Evolutionary Algorithms (MOEAs)), which comprise a set of techniques to find multiple trade-off solutions using EAs. MOOEAs have gained wide interest and success in

solving multi-objective problems for several reasons.Two of the most prominent, according to [Bechikh et al., 2017], are: MOOEAs allow finding several members of the Pareto optimal set in a single run of the algorithm; MOOEAs are less susceptible to the shape of the Pareto front than other algorithms.

Research in the field of MOOEAs recedes at least into the early eighties, with the work of David Shaffer often referred to as one of the first relevant steps [Deb, 2001]. In 1984 he presented the Vector-Evaluated Genetic Algorithm (VEGA), a modified version of a canonical GA, which showed that it was possible to develop techniques based in GAs that were able to find multiple trade-off solutions [Schaffer, 1985].

Not much later, in 1989, David Goldberg devised the use of the concept of *domination* in a modified GA [Goldberg, 1989], which led to further development by numerous researchers across the globe to date, leading to several implementations of MOOEAs [Konak et al., 2006], of which MOGA [Fonseca and Fleming, 1993], NPGA [Horn et al., 1994], NSGA, SPEA, NSGA-II [Deb et al., 2002] and MOEA/D [Zhang and Li, 2007] are a few well known examples. Using MOOEAs classification presented in [Bechikh et al., 2017, pp. 26–27], the first three can be classified as first generation algorithms (they were developed before 1995 and do not use elitism), the next two are classified as second generation algorithms (they were developed between 1995 and 2002 and use some form of elitism) and the latter is a third generation algorithm (these are algorithms developed after 2002 that use an approach concerned with the diversity along the Pareto front).

It should be noted that multi-objective optimization methods are sometimes classified under two main categories: weighted (or aggregated) approaches and Pareto-based approaches [Goh and Li, 2002]. However, when several objective functions are aggregated into a single one the algorithm handles only one function, so the optimization is performed without awareness of possible multiple trade-off solutions. This leads to these methodologies being classified by some authors as being of a single objective nature instead of a true multi-objective one.

In [Deb, 2001], K. Deb discusses this issue regarding the classification of these methodologies and notes that although single-objective optimization can be considered a degenerate case of multi-objective optimization, the latter is not a simple extension of the former. Furthermore, Deb notes, when an algorithm for single-objective optimization is used in a multi-objective optimization problem (with an aggregated function) it ignores a fundamental difference between the two approaches: in multi-objective optimization there is no single optimum solution, so (possibly) many optimal solutions are important and cannot be discarded, as a single-objective optimization algorithm would do since it produces only a single solution (and discards many other equally optimal along the way).

In EDA, usage of MOOEAs can be found in several fields, namely in generation and optimization of IC layout [Martins et al., 2012, Rocha et al., 2014, Martins et al., 2017] and, occasionally, in optimization of analog circuits, although not in the topology synthesis task;

it has been used in the component values optimization task, as in [Goh and Li, 2002] where, despite the title of the paper, the authors do multi-objective optimization, by the use of NSGA-II, to optimize component values of an OTA, on an already defined circuit topology.

It is worth referring that what sometimes may seem a multi-objective situation in a circuit synthesis problem is more likely just a single-objective problem with a constraint or, at least, is a problem that can be handled using a single-objective function with a constraint [Deb, 2001, pp. 126–133] without loss of significance. An example of this situation is how power consumption should be considered in some problems of circuit synthesis.

For instance, consider the evolved circuit depicted in Figure 2.10 : this circuit was synthesized by a GA which used as objective function the intended logic table for a NAND logic gate. In the execution of the GA, the variant NGSPICE [SourceForge, 2019] of the

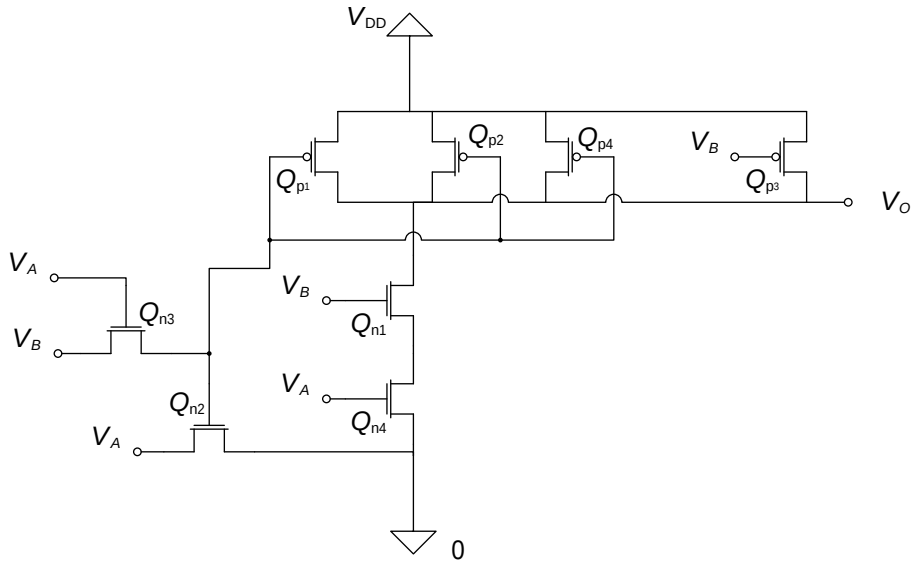


Figure 2.10: Evolved NAND circuit.

SPICE simulator was used to evaluate the fitness of each chromosome using level 1 model for the MOS transistors. The best circuit synthesized by the GA is the one presented in Figure 2.10 and in fact it fulfills the table of truth of a NAND gate according to the simulator. But it presents a major drawback: when both inputs have logic values '1', *i.e.*, when $V_A = V_B = V_{DD}$, input A drains much more current (through Q_{n2} to ground) than is necessary (and acceptable). This issue could be addressed with a multi-objective variant of a GA using two objective functions: one function implementing the truth table for a NAND gate and another function "measuring" the input currents. However this would be missing the point: the input current is not a goal of the problem in itself, rather it works as a threshold of feasibility. This comes from considering that all circuits that drain less current than a desired threshold are acceptable (or feasible regarding the matter of the order of magnitude of input currents), and all other circuits are not acceptable (or are considered unfeasible) because they would drain too much current from whatever sources drive this gate. So this issue would be better addressed if current drained from inputs is considered a

constraint rather than an objective. Similar issues, related to excessive current drainage from signal input sources or from power supply sources, may occur in analog synthesis problems [Forstén, 2012] if circuits are evolved without power consumption awareness somehow incorporated in the GA.

2.2.1.3 Fitness evaluation by circuit simulation

To evaluate the fitness of a chromosome that codifies a circuit, one or more simulations in SPICE (or in one of its numerous variants) are often performed. In real problems, it is very common that most of the computational time used in the synthesis of a circuit is taken by circuit simulations, so it is relevant to try to shorten the duration of these simulations.

In fact, what is commonly called "simulation time" is actually "fitness evaluation time", and usually includes successful and unsuccessful simulations, the latter often being caused by "poorly formed" netlists or by simulations with convergence issues.

Sometimes it happens that internal SPICE mathematical algorithms take a long time to converge or they don't converge at all, making it impossible to obtain a solution, at least in a timely, useful manner. So, these simulation attempts are discarded, even though the computational time has already elapsed.

It is not straightforward to predict convergence issues in SPICE just by analyzing what seems to be a properly devised netlist, so their occurrence is a known problem [Sapargaliyev and Kalganova, 2010b] that has to be dealt with by the circuit synthesizing program. This program should be able to take appropriate action when either a simulation takes too long or a simulation convergence error is issued by SPICE.

In order to reduce the possibility of a netlist being rejected by SPICE, it is quite common to preprocess it after the chromosome has been decoded (which is when a preliminary netlist is generated), but before it is actually submitted to SPICE for simulation [Koza et al., 1997c].

One procedure that often takes place in this preprocessing stage is eradication of nodes for which there is no DC path to ground; and another one is removal of dangling components. To eradicate nodes for which there is no DC path to ground, a large valued resistance is often inserted between ground and that node [Koza et al., 1997c]. And removal of dangling components is usually accomplished by spotting nodes in the netlist that have only one connection, followed by deletion of the dangling component (this procedure must be repeated until there isn't any node with only one connection, since deleting one component may produce another dangling component).

Preprocessing a netlist is frequently performed not only to reduce the possibility of a netlist being rejected by SPICE but also to try to reduce simulation time. Although it is not easy to calculate how the duration of the simulation varies with the number of nodes in a netlist, it is reasonable to assume that, as a rule, simulation time increases with that number.

In [Koza et al., 1997c] it is estimated that simulation time increases in an approximately sub-quadratic to quartic way. So, in order to accelerate simulations in SPICE, the number of nodes in the netlist should be minimized, if possible. Some procedures often referred in the literature to accomplish this goal include (but are not restricted to) substitution of all series and parallel compositions of passive components by a single component [Koza et al., 1997c].

Another way of reducing the number of nodes in a netlist is removal of isolated subcircuits, which lowers the simulation time by the double effect of simultaneously reducing the number of nodes and components.

The following schema summarizes these considerations:

- Reduction of time spent in fitness evaluation
  - "Faulty" simulations
    - Convergence issues (hard to predict).
    - Nodes without DC path to ground (use preprocess).
    - Dangling components (use preprocess).
  - "Clean" simulations
    - Reduction of netlist nodes (use preprocess).
    - Elimination of isolated subcircuits (use preprocess).
    - Simulation time too long (hard to predict).

Preprocessing a netlist before submitting it to SPICE is a common procedure in circuit synthesis software that relies in circuit simulation to evaluate the fitness of a chromosome but this procedure also consumes time and the balance is not always positive. Therefore, the amount of preprocessing to use is a topic always in discussion and some authors try to aim their development efforts to variants of the Holland's algorithm that do not use (or use a minimum of) netlist preprocessing [Sapargaliyev and Kalganova, 2010b].

2.2.1.4 Fitness evaluation by circuit simulation: what analysis?

Part of the design difficulty of amplifiers, namely operational amplifiers (that are usually intended to operate with negative feedback), is to optimize the location of poles and zeros of the transfer function so that intended specifications like open-loop gain, closed loop bandwidth and closed loop stability are achieved. In some published works this optimization is accomplished doing time-domain automatic analysis [Tavares et al., 2003, Santos-Tavares et al., 2008, Héctor et al., 2010], and GAs may be used in this optimization.

Time-domain automatic analysis, frequently done using a SPICE like circuit simulator, is also preferred by some authors over a usually faster DC analysis [Mydlowec and Koza, 2000, Sapargaliyev and Kalganova, 2010b]. In [Mydlowec and Koza, 2000], Mydlowec and Koza expose work where some computational circuits (squaring, square root and multiplier) are synthesized using GP, and they claim that however "[...]DC sweeps have the advantage of being considerably less time-consuming than time-domain simulations." they do not necessarily "[...]lead to robust circuits that correctly perform the desired mathematical function over time." [Mydlowec and Koza, 2000]. The work presented in this paper can be seen as a development of their previous work presented in [Koza et al., 1997b],

where another set of computational circuits were synthesized but, at that time, the fitness evaluation was accomplished by DC analysis (according to their definition, computational circuits are "[...]Analog electrical circuits that perform mathematical functions[...]"). And in [Sapargaliyev and Kalganova, 2010b], the authors also apply transient analysis to evaluate some computational circuits (squaring, square root, cubing and cubic root) as a way to maximize the number of valid chromosomes in each population "Due to the tolerance that transient analysis express to the same circuits that under DC analysis are treated as unconvergent,[...]".

2.2.1.5 Finishing the algorithm

GAs are iterative algorithms, in which each iteration produces a new population, called a *generation*, which will have fitter individuals (chromosomes) than the previous one. As seen in the diagram of Figure 2.9, the next stage of the algorithm is the decision as to whether or not it is time to stop iterating. Naturally the algorithm must be interrupted if some form of stop criterion is met. Common criteria to interrupt the algorithm are:

1) achievement of a fitness threshold by at least one chromosome;
2) maximum number of iterations exceeded;
3) No improvement in the last N iterations;
4) execution time too long.

These are common criteria but many other criteria are also used.

2.2.1.6 Selection

In the next stage of the algorithm, referred as *selection* in Figure 2.9 but sometimes called *reproduction* in the literature, the (already evaluated) population is evolved by a *selection* technique, in which there is a higher probability of the fittest chromosomes to be selected to contribute to the next generation. Other reproduction operators will later be applied to these chosen chromosomes, producing the next generation. In some variants of the algorithm, a few of these chosen chromosomes are simply copied to the next generation (in a process called *elitism*). Selection and other reproduction operators maintain a constant population size, which implies that in GAs all generations have the same number of chromosomes (this is the rule for most implementations of GAs, although some variants may change population size over time [Sripramong and Toumazou, 2002]).

The selection procedure itself does not create any new sample point in the search space: rather, it is responsible for the *survival of the fittest* characteristic of GAs, and has an "exploitation" effect as it reduces diversity in the population. There are several selection techniques used in GAs [Whitley et al., 1989, Goldberg and Deb, 1991, Whitley, 1994, Tomassini, 1995] and it is quite possible that new techniques will continue to appear as long as GAs are used and enhanced.

2.2.1.7 Crossover and mutation

The most common reproduction operators (other than *selection*) are *crossover* and *mutation*, and both have numerous variants. The *crossover* operator uses genetic material from two chromosomes to generate two new ones, by combining segments of their parents. *Crossover* occurs only with some probability so not all selected chromosomes generate some offspring. The role of crossover is to create new individuals recombining (hopefully good) genetic material of others, thus sampling new points in the variable space.

After the crossover operator is applied, the mutation operator is executed for every chromosome with some probability, usually low. If executed, it changes one or more genes of that chromosome. The role of mutation is to keep diversity in the population, restoring unexplored or lost genetic material into the population in order to widen the search area and to prevent premature convergence to suboptimal solutions.

When the codification of a chromosome is a string of bits, as in GA-c, the crossover and mutation operators are applied in a bit-oriented way. For the crossover operator this usually means swapping fragments of binary strings. And for the mutation operator this usually means that a random bit (or a few bits) is (are) flipped with some probability. But when the codification of a chromosome is accomplished with real numbers (usually referred as *real coding* as opposed to *binary coding*), which is done in many variants of GA-c, these operators are no longer applied in a bit-oriented way. Rather, they are applied using several possible functions [Goldberg, 1989, pp. 80–85] [Deb et al., 2007, Gaffney et al., 2010, Yoon and Kim, 2012] that generate offspring by changing one or more gene's values of a chromosome (a gene's value is the value of one of the variables encoded in a chromosome).

It is well known that mutation and crossover probabilities significantly affect the behavior and the performance of the GA-c [Zhang et al., 2007], so several authors use their own method to calculate them. For example, in [Zhang et al., 2007] fuzzy logic is used to adaptively adjust mutation and crossover probabilities. And in [Barros et al., 2010] a dynamic mutation operator, equipped with an *undo* feature, is used that evaluates whether the mutated chromosome has a better fitness, *accepting* the new chromosome if it does but also *accepting* it with some probability if it does not. If it is not accepted, the operator activates the *undo* feature, *undoing* the mutation. That probability is adjusted in a simulated annealing fashion, which means that there is a higher probability of accepting fitness worsening in the early iterations of the algorithm than in later iterations, promoting wider search exploration first and local exploration later.

The terms *exploitation* and *exploration* have their own and autonomous meanings in the English language, and some authors, such as Deb and Goldberg, use them to represent related concepts in the context of GAs. They relate the concept of *exploitation* with the loss of diversity that can occur in the GA due to too much selection pressure eventually created by the selection operator (and also due to excessive elitism), and the concept of *exploration* with the enhancement of population diversity that comes from the exploratory

effect inherent in the mutation and crossover operators [Goldberg and Deb, 1991][Deb, 2001, pp. 93,100,116,414]. A good balance between the reduction and enhancement of population diversity will allow a GA to have an adequate search property [Deb, 2001, pp. 116].

2.2.1.8 Iterate to the next generation

The new generation obtained after the execution of the mutation operator must be evaluated, so the process is going to be repeated, as shown in the diagram in Figure 2.9. This is the end of an iteration and another will start so, if there is a generation counter in the algorithm implementation, this is usually the moment (after mutation and before fitness evaluation) to update it.

2.2.2 Variable-length chromosomes

The extension of the GA proposed by Holland [Holland, 1975] with new concepts probably began soon after the publication of Holland's work in 1975, and indeed there are numerous published works in this broad area. One class of extensions to the GA-c often referred to be of interest in the field of circuit synthesis is the concept of a VLC. This concept is a model that has been around for quite some time [Kajitani et al., 1996, Iwata et al., 1996, Zebulum et al., 1998, Lohn and Colombano, 1998], and has been applied in several scientific fields [Kim and de Weck, 2005, Brie and Morignot, 2005, Arora and Sinha, 2013, Deif and Gadallah, 2014, Pawar and Bichkar, 2015, Ni et al., 2016].

In GAs that use FLCs, all encoded candidate solutions in the algorithm are equal in length, *i.e.*, all chromosomes have the same number of genes. But in many problems it may be impossible to know a priori which chromosome length is needed in order to be able to codify one optimum solution. So, the execution of the algorithm may begin with chromosomes with some length that is later adjusted during the evolution of the population. One example of this difficulty of predicting which is the right chromosome length to codify one optimum solution is what happens in some circuit synthesis problems where the number of components (and nodes) in the final circuit is unknown a priori.

The use of VLCs in circuit synthesis tools is known since, at least, the works of John Koza [Koza et al., 1992, Koza et al., 1997a, Koza et al., 1997b, Koza et al., 1997c]. In these works, based in GP, the depth of the trees used to represent the circuits define the length of the associated chromosome. In other works not based in GP but that follow GA more closely, VLCs are also used, as is in [Kajitani et al., 1996, Lohn and Colombano, 1998, Lohn et al., 2000, Sapargaliyev and Kalganova, 2010b]. VLCs are also mentioned in works that aim circuit synthesis but are not based in GAs. For eg. Yuan [Yuan and He, 2010] used VLCs to synthesize DC amplifiers (DC-coupled amplifiers) based on a differential evolution algorithm [Storn and Price, 1997].

VLCs poses some questions to the crossover and mutation operators, so these operators are often adapted [Cavill et al., 2006, Hutt and Warwick, 2007, Stringer, 2007, Deif and

Gadallah, 2014, Pawar and Bichkar, 2015] to work in populations with chromosomes that don't have a constant length.

Even though VLCs are commonly referred in circuit synthesis related works, they are far from being an all-purpose panacea and in fact there are still many published works about circuit synthesis that do not use VLCs and yet implement other extensions to Holland's GA. One such example is the work presented in [Karci et al., 2016] where the authors present an extension to the GA that improves its local search capability applied to analog circuit synthesis. In this work, which doesn't use VLCs, a procedure to improve the weak local search capacity of the GA is proposed, in which a *speciation* algorithm is introduced. Basically, this *speciation* algorithm reinitializes the population if the best fitness in the population falls beyond some threshold during evolution. This reinitialization is accomplished by copying the best chromosomes to the rest of the population with only minor changes (either in their electrical connections or in their values).

2.3 Automatic synthesis of amplifier circuits

In ADA, automatic synthesis of amplifier circuits has been a research topic for several years, and several published works have pursued the generation of amplifier circuits employing EA-based methods. In [McConaghy et al., 2009], very interesting results were obtained using multi-objective selection (NSGA-II [Deb et al., 2002]) and other techniques (GP [Koza et al., 1992], Age-Layered Population Structure (ALPS) [Hornby, 2006]), while using a library of building blocks (among other types of components that can be used to generate a circuit). In [Sripramong and Toumazou, 2002], several amplifier topologies are generated with very interesting specifications, using GP and some additional techniques, like a current-flow analysis that is a procedure used to identify and correct any faulty design prior to simulation. Although single components (like transistors, resistors and capacitors) are used to build the circuit, they also use a user-defined library of building blocks.

In [Trefzer, 2006], circuit synthesis is performed at flat-level (i. e. at device-level, or as the author says, "from scratch"), but using only transistors, available on an Field Programmable Transistor Array (FPTA). This FPTA is used in-the-loop (i. e. inside the evolutionary loop) to evaluate the generated circuits. More than one EA is used (a GA, a variant of a GA, and a MOEA [Deb, 2001]), and the results are compared for the generation of logic gates, comparators, square-wave oscillators, and very simple opamps. The latter resemble elementary versions of a differential stage of an opamp, although, as the author points out, "(...) it can be stated that, despite the EA did not discover any new groundbreaking design, it is still an impressive result that the algorithm achieved to synthesize a differential input stage (...) without prior analog design knowledge".

A classic work in the field of ADA dedicated to the synthesis of CMOS amplifiers with GAs is [Kruiskamp, 1995], where, as is typical for GAs, topology selection and circuit

sizing are performed simultaneously. Although some very promising results have been obtained, including not-so-simple amplifier topologies, the generation is entirely based on building blocks, yielding results very much based on well-known classical topologies. In [Zebulum et al., 2000], several techniques employing VLCs (also called Variable-length Representations (VLRs)) in GAs are used to generate circuits at flat-level without any use of building blocks. Several case studies are presented, namely for the synthesis of combinational digital circuits (such as multiplexers, comparators and parity functions) and passive filters (using resistors, capacitors, and inductors). However, only one result about amplifiers is shown, an amplifier made of 7 transistors (BJTs of NPN and PNP type with unknown characteristics) and although the authors claim a gain of 55 dB, nothing is mentioned about linearity, output offset, dynamic range or bandwidth.

Some authors use EAs that are not GAs to generate amplifiers, as in [Yuan and He, 2010], which uses a variant of the Differential Evolution (DE) technique proposed in [Storn and Price, 1997]. The original DE technique uses FLCs, but the authors of [Yuan and He, 2010] adapted it to use VLCs and presented one amplifier with BJTs and resistors without using building blocks. This amplifier is claimed to have a DC gain of 63.16 dB, an output DC bias of 5.04 V, a power dissipation of 9.34 W, and a 3 dB bandwidth of 458.8 kHz. However, these characteristics are obtained when the amplifier is used in a negative feedback shunt-shunt topology [Sedra and Smith, 2014], which linearizes the overall system response and increases its bandwidth. This is possible because the amplifier uses an inverter topology, which results in negative gain, and allows the creation of a negative feedback topology by using a two-resistor feedback network. Thus, it is not clear what are the open-loop characteristics of the amplifier, namely nothing is said about its linearity and bandwidth.

Another application of the DE technique and some of its variants [Kennedy and Eberhart, 1995, Karaboga and Basturk, 2008] is presented in [Sabat et al., 2009], where a transconductance operational amplifier is fully sized and the performance of three EAs is compared. In this study, the selection of the topology is left aside and is done beforehand by a human designer, but the results are interesting because they show that even in isolation, the problem of sizing is in itself a difficult one, since not all the algorithms tested succeeded in all the specifications desired for the amplifier.

In [Wang et al., 2008], a combination of the divide-and-conquer approach [Torresen, 1998] and GP, along with a proposed new circuit representation method based on a two-layer evolutionary scheme, is used to generate circuits without building blocks. The authors refer that the embryo circuit is given as a primitive individual of the circuit, but do not present an example of such a circuit and are not clear about its randomness (or about the relationship, if any, between the primitive individual and the generated circuit). The results presented show an amplifier based on an ideal opamp, in a topology that uses both negative and positive feedback achieved by resistor networks that have been synthesized by the algorithm. However, the active components of the amplifier are "buried" inside the ideal opamp, which is in fact a kind of building block. Another circuit shown is called a low-pass

filter by the authors, but its frequency response is that of a band-pass filter. The circuit consists of two amplifier stages, each with an NPN transistor, the first in a degenerate common emitter topology and the second in a common collector topology. However, it is not clear what exactly has been generated by the algorithm, although the achieved gain (20 dB) and high frequency cut-off (30 kHz) correspond to the desired (objective) values.

2.4 Conclusions

A relevant issue of the research hypothesis initially raised in this work concerns the ability of a GAs (namely its canonical version, GA-c) to generate useful circuit topologies and, eventually, new ones. This question remains largely unanswered in published work (known to the author).

From early research works in the area of circuit synthesis with GAs until present time there was an enormous evolution in computer power, but it is not clear how this affects the results that may be expected nowadays, and why some lines of work are no longer followed despite the computer power available. There seems to have been an initial enthusiasm that later faded, probably because the results were no longer the desired ones at some point along the way. But how exhausting was the research effort in each direction? What made some research paths fade away? What kind of circuits cannot be synthesized by a GA? What are the expected obstacles in automatic synthesis of analog and digital circuits? Is synthesis at flat circuit-level a reasonable research path to follow or is it a chimera? Many questions are still only partially answered.

What underlies these issues is that there still seems to be insufficient certainty about the origin of the GA limitations in circuit synthesis: is it a unsurpassable limitation of the algorithm itself or is it just a limitation resulting from computer power still insufficient to produce the results we aspire to?

This work seeks to provide some contribution to answer some of these questions.

CHAPTER 3

Circuit Synthesis with a Genetic Algorithm

This chapter describes the application of a GA in circuit synthesis, illustrated by some results and presenting some techniques that deviate the implemented version of the GA from the original version, i.e.*, from GA-c.*

3.1 Implementing and testing a genetic algorithm for circuit synthesis

A relevant issue of the research hypothesis initially raised in this work concerns the ability of GAs (namely its canonical version, GA-c) to generate useful circuit topologies and, eventually, new ones, which is a question that remains largely unanswered in published work (known to the author).

GAs are search and optimization algorithms which have been tested both for topology generation and circuit optimization, as mentioned in Section 2.2. However, it is for circuits' optimization that more applications are found that actually use GAs; its commercial use for synthesis is not yet common.

This chapter describes the application of a GA to circuit synthesis, starting with the use of GA-c and then progressing to variants that were considered to be better suited to this problem. It will not be described here what a GA is, but how one was applied to the problem of circuit synthesis.

The chain of actions executed by the GA used to solve this problem follows the flowchart

already presented in Figure 2.9 but parallelism was used in some stages of the algorithm, namely in the fitness evaluation stage, which is surely the most time-consuming stage of the algorithm in the context of circuit synthesis. As there was capability to parallelize code execution, parallelization was extended to other stages of the algorithm that are asynchronous by nature, namely to the crossing and mutation stages, so a flowchart that better illustrates the GA's chain of actions is presented in Figure 3.1.

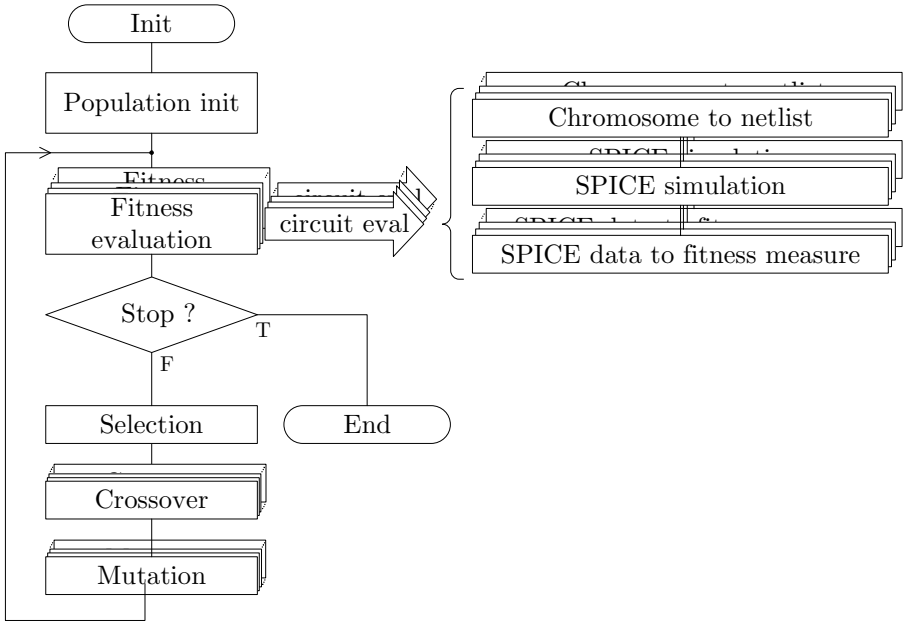


Figure 3.1: Paralelism and fitness evaluation in the genetic algorithm.

In the fitness evaluation stage several instances of the circuit simulator are executed simultaneously, using the default load distribution policy of a Linux operating system which ends up running those instances in a basis of one per core (because the GA is configured to use as many threads in fitness evaluation as available cores reported by the operating system).

3.1.1 Using multi-threaded parallelism

GAs are highly parallelizable algorithms and there are several parallelization models that can be adopted for GAs implementation [Whitley, 1994][Haupt and Haupt, 2004, pp. 137–145]. In this work preliminary tests were performed to assess the suitability of the *boss-worker* model and later its *thread pool* [Bradford et al., 1996, pp. 31–37] variant was selected to run the implemented GA.

To use thread-level parallelism, programs must have a multi-threaded design. Programming in a multi-thread model is highly prone to errors that are not easy to debug and that are not found in a program that does not use any form of parallelism, such as race conditions and deadlocks [Bradford et al., 1996, pp. 40–41, pp. 193–194], hence a potentially heavy debug burden is normally expected.

To minimize the propagation of multi-threaded-related programming bugs to other stages

of this work several tests were performed to achieve a satisfactory degree of confidence in the developed software regarding the multi-thread implementation [Campilho-Gomes, 2014]. As an outcome of these tests the multi-thread model to be used in the GA was chosen and tested. Also, some important insight in the parametrization used to configure the multi-thread implemented model was acquired and used later in the course of this work.

3.1.2 Testing the developed program

As in many other cases where a program is created to comply with a certain algorithm, developing a program to implement a GA entails its own uncertainties, namely those related to the correctness of the interpretation of the algorithm's description which, despite the abundant literature on the subject, always has some degree of incomplete characterization of one or another step of the algorithm (or of its variants). Even if the description of the algorithm, or the variant implemented, is clear, complete, and free of ambiguities, choices and decisions must be done regarding the concrete actual implementation and parametrization of the algorithm. And, anyway, programs have bugs. So, the doubt if the program is really implementing a GA is, probably, a reasonable doubt, that emerges from time to time in the endeavor of maintaining and evolving such a computer program, which is inherent to a research work like this one. Therefore, throughout this work the GA was tested several times with a set of benchmarking functions as a way of verifying that the developed program still retained the search and optimization capabilities that are expected from an GA. It is, in the end, a way to increase confidence that what is actually implemented by the program is, probably, a GA. This set of benchmarking functions includes a subset of single-objective unconstrained optimization functions like Sphere function, Rastrigin function, Ackley function, Rosenbrock function, Booth function, and Himmelblau function [Adorio and Diliman, 2005, Jamil and Yang, 2013]. And it also includes another subset of functions devised by the author that involve the synthesis of simple or previously synthesized circuits.

3.2 Fixed-length chromosomes

A common characteristic to GA-c and to numerous tests performed in the course of this work is the usage of FLCs. As referred in Chapter 2, this coding scheme uses a constant number of genes in each chromosome throughout every run of the GA. These genes can use bit strings as gene descriptors (like in GA-c) or real variables, as many variants of GA-c use.

In circuit synthesis, the use of FLCs means that all components that are necessary to generate a circuit must be present in the chromosome since the first generation of the GA. In some problems this may be desirable since it works as a constraint to the number (and type) of components the GA can use to solve a problem. In other problems it may be an undesirable challenge to decide *a priori* how many components and of what type should be present in the first generation: if too few are provided (too few in total or even too few of

a particular type) it may be impossible for the GA to solve the problem; and if too many are provided it may be extremely time consuming to simulate circuits that have many superfluous components.

This section describes the approach that was made to the synthesis of circuits with a GA that uses FLCs.

3.2.1 Testing the genetic algorithm in circuit optimization

The design of integrated analog circuits typically has three distinct stages: choice of circuit topology, dimensioning of components, and layout design. The automatic synthesis to be studied in the context of this work, carried out by a GA, focuses on the first two stages. To evaluate the algorithm's performance in executing the second stage (component sizing in a circuit), tests were carried out in which the execution of the first stage was suppressed. In these tests, the circuit topology is initially chosen, and the GA is then left with the task of sizing some components so that the circuit meets a set of specifications.

3.2.1.1 Optimization of a two-stage differential amplifier

In the test described in this section, it is intended that the GA sizes some parameters of the components of a two-stage differential amplifier in order to optimize its performance according to a set of specifications.

The amplifier circuit is shown in Figure 3.2, in which parameter W of the transistors must be sized by the GA so that the desired specifications are met. This circuit is based in [Sedra and Smith, 2007, pp. 749–752] and the desired specifications are summarized as follows:

- Intended DC gain of 1109, that is, 60.9dB.
- All transistors have the same L but their W is modifiable by the GA.
- Some transistors are already paired, that is, its W is necessarily the same in the following cases: $W_{Q_1} = W_{Q_2}$, $W_{Q_3} = W_{Q_4}$, $W_{Q_5} = W_{Q_7} = W_{Q_8}$.
- In order to approximate the models of the transistors used by NGSPICE to the characteristics of the transistors used in [Sedra and Smith, 2007, pp. 749–752], the models used in simulation were as follows:

```
.MODEL MY_PMOS PMOS level=1 VTO=-0.8 KP=40e-6 LAMBDA=0.1
.MODEL MY_NMOS NMOS level=1 VTO=0.7 KP=160e-6 LAMBDA=0.1
```

- Current source I_{REF} is 90 µA
- All circuit simulations were performed with transient analysis, that is, they were performed with .TRAN commands in NGSPICE.

The embryo circuit for this test contains the circuit depicted in Figure 3.2 plus some extra components as shown in Figure 3.3. These extra components create the "context" circuit

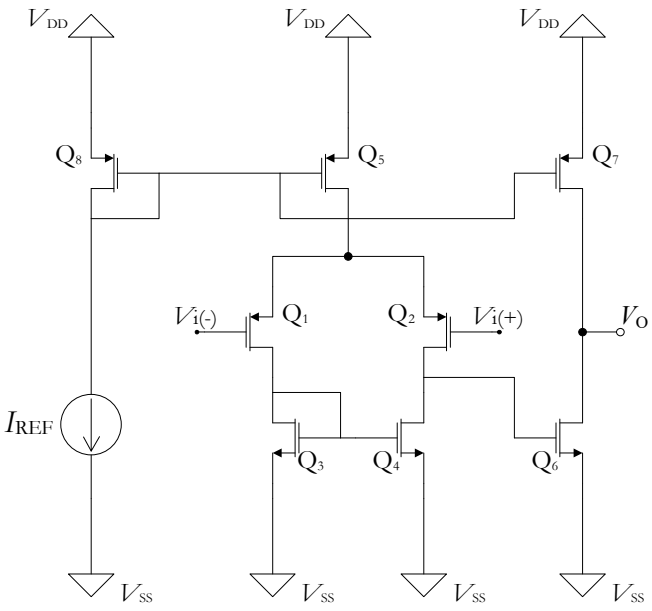


Figure 3.2: Two-stage differential amplifier circuit.

for the simulations that will be submitted by the GA to be performed by NGSPICE.

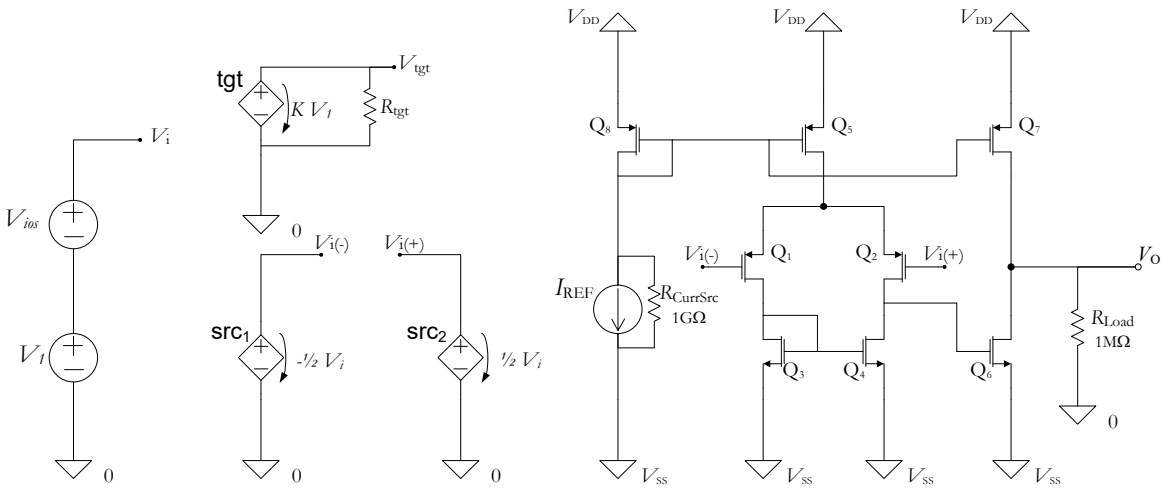
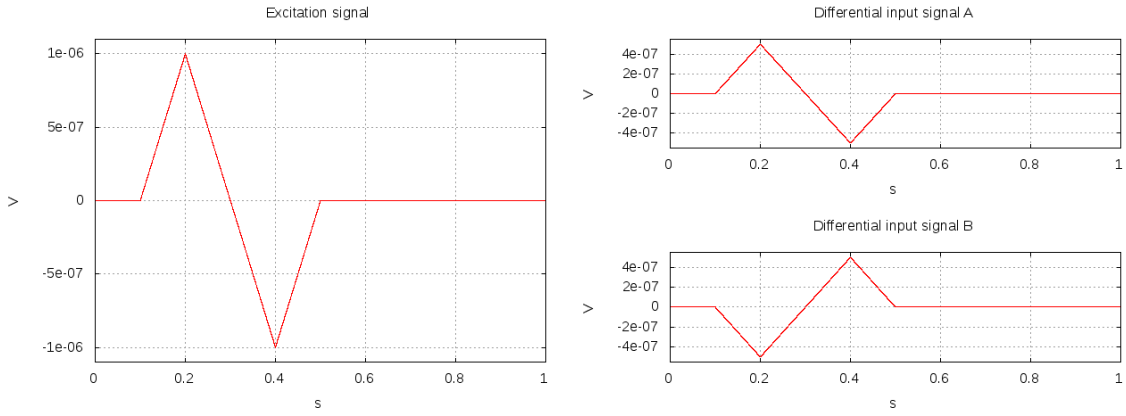


Figure 3.3: Context circuit for the two-stage differential amplifier.

Voltage source V_1 generates the circuit excitation signal, as shown in Figure 3.4. A quite small amplitude was used in several early tests (in this case it was 1 µV) as a way of trying to make the GA search effort to be in the most linear range of the amplifier. Also, this signal is "slow enough" (has a low frequency) so that the influence of the dynamic response of the circuit is negligible in this analysis (that is, so that the effects resulting from non-idealities of the amplifier, such as finite bandwidth and finite slew rate, can be neglected for now).

The signal from source V_1 is transformed into a pure differential excitation (*i.e.*, without a common mode component) by the auxiliary sources src_1 and src_2. These sources are the input sources of the differential pair (note that it was not a concern of this test to verify the response to common mode excitation since there was no specification regarding the

amplifier's CMRR).



(a) Amplifier excitation signal

(b) Differential input signals

Figure 3.4: Amplifier input signals.

In this test there was no specification regarding the output offset voltage but the eventual existence of such offset could interfere with the process used to calculate the fitness of each circuit. This fitness is evaluated by comparing the signal obtained at the amplifier's output with a desired (or target) signal produced by an auxiliary voltage source (as explained later). In order to try to minimize the interference of the amplifier's output offset in the evaluation of a circuit's fitness, an offset voltage source V_{ios} was added to the excitation signal V_1, as seen in Figure 3.3. This way, the GA is able to modify the input offset voltage of the amplifier (the value of this offset is one of the variables sized by the GA) and thereby modify its output offset voltage, eventually minimizing it.

Sizing results

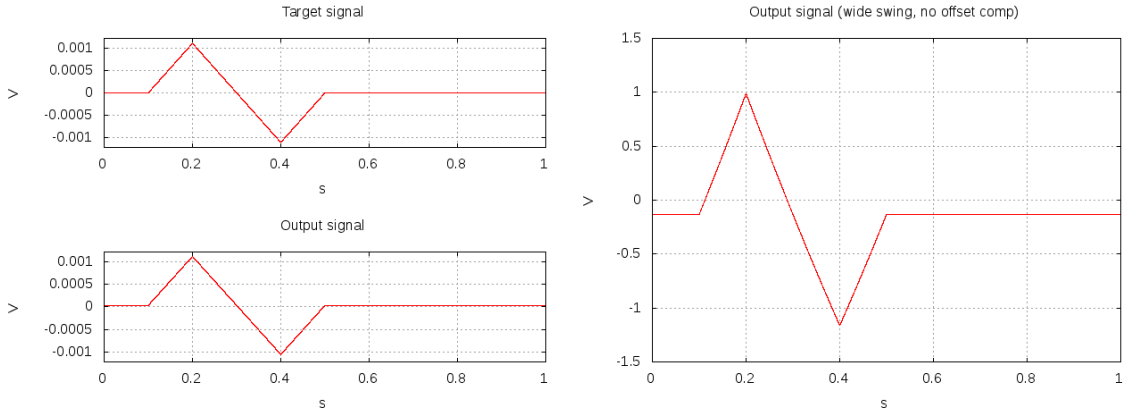
The type of GA used in this test is close to GA-c, so chromosomes are composed of bit strings where each gene has 16 bit linearly codifying the range of each variable (using a coding method sometimes called "multiparameter, mapped, fixed-point coding" [Goldberg, 1989, pp. 80–84]). Considering transistor pairing and input offset voltage source, the GA has only five variables to size:

- $W_{1,2}\left(= W_{Q_1} = W_{Q_2}\right)$
- $W_{3,4}\left(= W_{Q_3} = W_{Q_4}\right)$
- $W_{5,7,8}\left(= W_{Q_5} = W_{Q_7} = W_{Q_8}\right)$
- $W_6\left(= W_{Q_6}\right)$
- V_{ios}

Using random initialization for those variables, it was verified in numerous runs that the GA achieves *good* results almost always in less than 100 generations, considering *good* (or

successful) results those that present high similarity between the desired output signal and the signal that is actually achieved by a run.

In Figure 3.3, voltage source *tgt* generates signal V_{tgt} which is considered the desired signal at the output of the amplifier, given the input signal V_1. It is this signal V_{tgt} that is compared with the circuit's output V_o in each generation of the GA to establish a measure of the amplifier's achievement of the desired gain (the circuit's fitness). These signals are represented in Figure 3.5a for a successful execution of the GA. The gain of the circuit sized by the GA is approximately 1070 (60.6 dB) and the output signal has an offset voltage of about 34 µV. The GA sized the input offset voltage source to $V_{i_{os}} = 122\,\mu V$ which suggests that the amplifier has an output offset voltage of approximately -131 mV (when it is not compensated by $V_{i_{os}}$). In order to verify this output offset and the behavior of the circuit for signals of greater amplitude, the circuit was tested with null input offset and with an input signal similar to the one shown in Figure 3.4a but this time with amplitude 1 mV. The signal produced by the circuit is depicted in Figure 3.5b, which has an offset of approximately -132 mV and a voltage swing of approximately 2.14 V.



(a) Desired and achieved signals

(b) Wide output signal (with $V_{i_{os}} = 0$)

Figure 3.5: Target (desired), and output signals.

The fitness value of each chromosome is a function of the mean square error given by eq. 3.1 (where $T_0 = 1$ s), which is the objective function of this problem. As is known, the fitness function does not always coincide with the objective function [Deb, 2001, pp. 86–88] [Goldberg, 1989, pp. 75–76], which is the case here. That *rms* error is in fact determined by eq. 3.2, given the discrete nature of the data obtained in simulation, where N is the dimension of the vectors containing signals $V_o(n)$ and $V_{tgt}(n)$. The GA tries to minimize that error by maximizing the merit function given by eq. 3.3 which is the actual fitness function that evaluates each chromosome.

$$e_{rms} = \sqrt{\frac{1}{T_0}\int_0^{T_0}\left(V_o(t) - V_{tgt}(t)\right)^2} \tag{3.1}$$

$$e_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}\left(V_o(n) - V_{tgt}(n)\right)^2} \tag{3.2}$$

$$f_{fit} = \frac{1}{e^{e_{rms}}} \tag{3.3}$$

Figure 3.6 shows the netlist that describes the circuit to be optimized by the GA. It is this netlist that is submitted to NGSPICE but with the line containing the `.param` command edited, in which the strings `gene0`, `gene1`, ..., `gene4` are replaced by values from the chromosome being evaluated.

```
Two-Stage CMOS Op Amp
VDD 100 0   2.5V
VSS 200 0   -2.5V
.param W12=gene0 W34=gene1 W578=gene2 W6=gene3 Vin_offset=gene4
Rload       15      0   1000k
I1 16 200 DC 90E-6
RCurrSrc 16 200 1E9
* Dif pair
M1  13  11  10  10  MY_PMOS W={W12} L=0.8u
M2  14  12  10  10  MY_PMOS W={W12} L=0.8u
* Active load
M3  13  13  200 200 MY_NMOS W={W34} L=0.8u
M4  14  13  200 200 MY_NMOS W={W34} L=0.8u
* Dif pair current source
M5  10  16  100 100 MY_PMOS W={W578} L=0.8u
* Output stage
M7  15  16  100 100 MY_PMOS W={W578} L=0.8u
M6  15  14  200 200 MY_NMOS W={W6}   L=0.8u
* Curr source ref transistor
M8  16  16  100 100 MY_PMOS W={W578} L=0.8u
* Input sources
V1      300     0 DC 0 PWL (0 0 100m 0 200m 1u 400m -1u 500m 0 1000m 0)
Vios    1       300 {Vin_offset}
* Aux sources
Etgt        3   0       300     0       1109
Rtgt        3   0   1E3
Emeas1  4   0   13  14  1
Rmeas1  4   0   1E6
Esrc1       11  0       1       0       -0.5
Esrc2       12  0       1       0        0.5
* Models
.model MY_PMOS PMOS level=1 VTO=-0.8 KP=40e-6 LAMBDA=0.1
.model MY_NMOS NMOS level=1 VTO=0.7 KP=160e-6 LAMBDA=0.1
* Analysis
.tran 100u 1
.end
```

Figure 3.6: Spice netlist for the optimized circuit.

At the end of a successful run, the GA produced a netlist in which that line was updated and now reads:

`.param W12=0.000006 W34=0.000006 W578=0.000068 W6=0.000012 Vin_offset=0.00 0122`

That line shows that the five variables were sized by the GA to the following values:

- $W_{1,2} = 6\,\mu\text{m}\,(W/L = 7.5)$
- $W_{3,4} = 6\,\mu\text{m}\,(W/L = 7.5)$
- $W_{5,7,8} = 68\,\mu\text{m}\,(W/L = 85)$

- $W_6 = 12\,\mu m\,(W/L = 15)$
- $V_{ios} = 122\,\mu V$

Although in this test the GA only sized 5 variables, it was nonetheless possible to verify that the global procedure works, namely that the coding used is good enough for this type of problem. It was also verified the operability of the interaction between the GA and the NGSPICE circuit simulator, as well as the use of the transient analysis for this type of sizing.

As a precaution, several runs that were considered successful were later tested manually, using the netlist produced by the GA to simulate the sized circuit. These verification tests were carried out using NGSPICE in interactive mode and also in another SPICE-like simulator to compare results.

3.2.2 Tunning some parameters of the genetic algorithm

In several tests, not only while solving generic problems of optimization or with problems about circuit dimensioning but also in the resolution of other problems, it was noticed a great sensitivity of the speed of convergence of the GA with the seed of the random generator, that is, the evolution of the fitness function varied widely with the initial value of the genes of the first generation. Although there is more than one possible cause for this phenomenon, it is inevitable to consider the two most likely causes: the existence of bugs in the implementation of the algorithm and/or a deficient parameterization. As no bugs were found to justify this effect, attention was given to some parameters of the GA, namely, to the probabilities of crossing and mutation. In order to try to understand how those probabilities affect the convergence of the GA (in the implemented version), tests were made that systematically scan the (apparently) useful ranges of those probabilities. These results are presented in the following sections.

3.2.2.1 Testing with a third degree polynomial

The fitness function of these tests uses a third degree polynomial, given by eq.3.4 for which the GA is requested to find the coefficients that were used in the target polynomial, given by eq.3.5.

$$y = a_3 x^3 + a_2 x^2 + a_1 x + a_0 \tag{3.4}$$

$$y_{tgt} = a_{3_{tgt}} x^3 + a_{2_{tgt}} x^2 + a_{1_{tgt}} x + a_{0_{tgt}} \tag{3.5}$$

To determine the fitness value of each chromosome, a function was used that calculates the mean square error, given by eq. 3.6 where N is the dimension of the vectors that contain the values that result from sampling those polynomials (with N points equally spaced in a range of $\mathbb{R}$ that includes all the roots of y_{tgt}).

$$e_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}\left(y(n) - y_{tgt}(n)\right)^2} \tag{3.6}$$

It is the minimization of this error that is sought with the GA, using eq. 3.7 as the fitness function.

$$f_{fit} = \frac{1}{e^{e_{rms}}} \tag{3.7}$$

Tests with constant mutation probability

In this set of tests, the mutation probability, the crossover probability and the random generator seed were varied. The probability of mutation was varied between 0.01 and 0.32 in a geometric sequence, with the following values being tested:

$$p_{mut} \in \{0.01, 0.02, 0.04, 0.08, 0.16, 0.32\}.$$

The probability of crossover was varied between 0.3 and 0.9 in a linear sequence, with the following values being tested:

$$p_{cross} \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}.$$

For the random generator seed, three different values were always tested, which makes a total of $6 \times 7 \times 3 = 126$ runs of the GA. Data produced by these tests is shown in graphical form in Appendix A and is summarized in Figures 3.7 and 3.8 which show the fitness of the best chromosome in the run (Figure 3.7) and the average fitness in the population obtained in the last iteration (Figure 3.8).

Note that the aforementioned probability p_{mutat} represents the probability of mutation for each chromosome, which results in a relative frequency of chromosomes in the population that have undergone some mutation which, in average, has the same value as p_{mutat}. When the GA uses bit strings as a codification scheme, like GA-c and this test use, the probability that an N_{bit} chromosome will mutate is given by $p_{mutat} = 1 - (1 - p_{bit})^N$, where p_{bit} is the mutation probability for each bit of the chromosome. That is, to obtain a certain relative frequency of mutation in a population, the p_{bit} probability must be calculated as a function of the number of bits of the chromosome. So, if the chromosome size is adjusted (in a bit string codification scheme), there may be an interest in adjusting p_{bit} in order to keep p_{mutat} constant.

In all tests referred above the mutation probability was constant during each run of the GA but during those tests there was a growing interest in evaluating what would happen if the mutation probability varied, namely starting with a higher value at the beginning of the run and then decreasing as the run approached the end. The idea was to provide the GA with a high ability for space exploration at the beginning of the execution that would shift smoothly to a fine-tuning ability at the end of the execution.

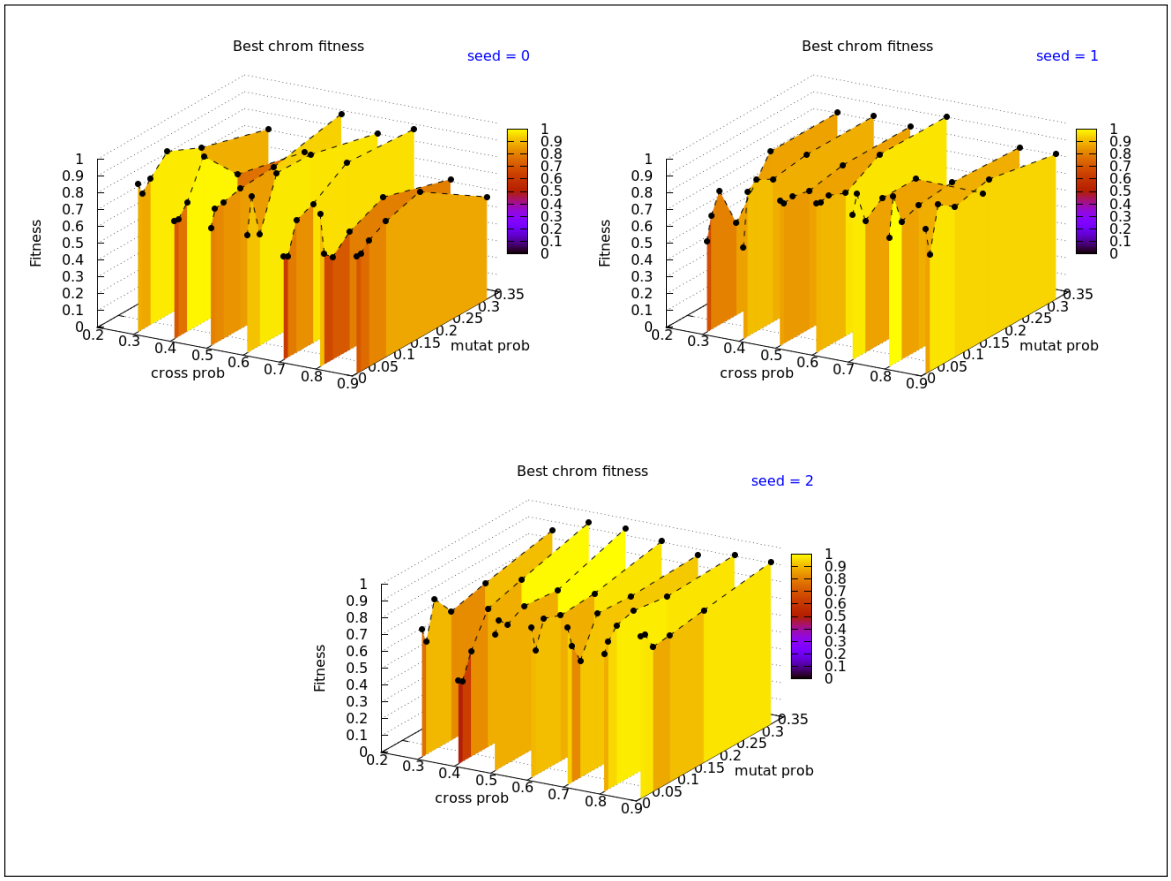


Figure 3.7: Best chromossome fitness in tests with constant mutation

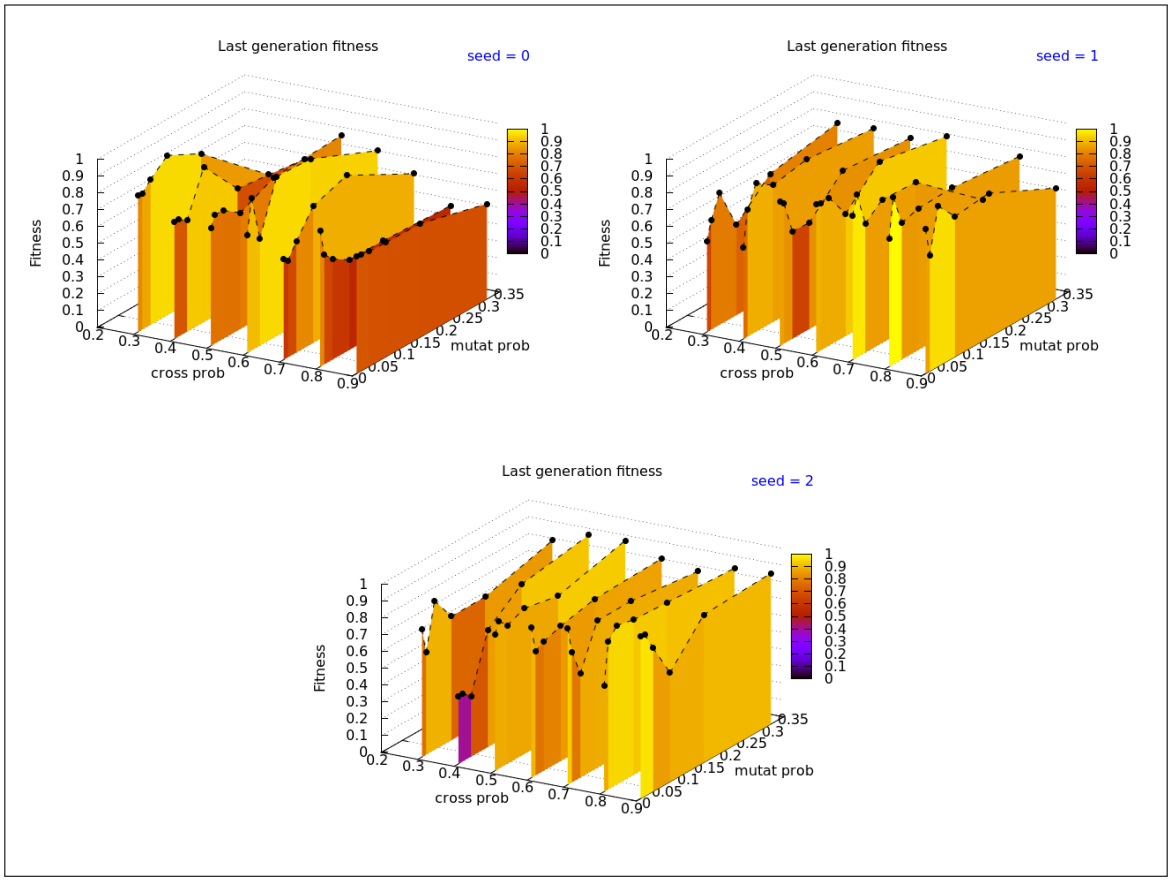


Figure 3.8: Population average fitness in tests with constant mutation

Tests with variable mutation probability

In this set of tests the mutation probability did not remain constant throughout each test, as it decreased linearly towards zero over the N generations of the run, leaving the GA with some characteristics of a simulated annealing process. The initial value of this probability was established by sweeping the same set of 6 values used in the previous tests. It is this initial value that is marked in the graphs that follow.

Similarly, the crossover probability was established by sweeping the same set of 7 values used in the previous tests and each pair of p_{mutat} and p_{cross} was run for 3 seeds, leading (again) to a total of $6 \times 7 \times 3 = 126$ runs of the GA. As before, data produced by these tests is shown in graphical form in Appendix A and is summarized in Figures 3.9 and 3.10.

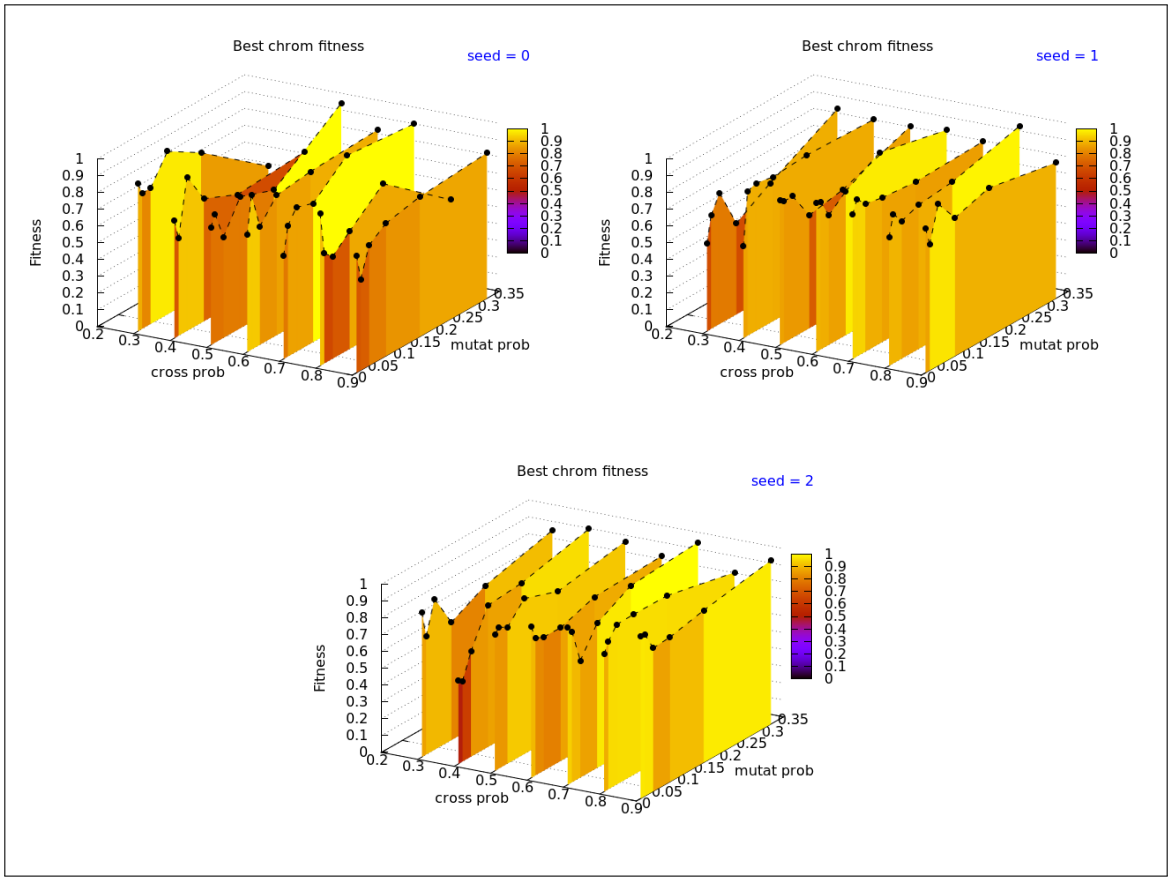


Figure 3.9: Best chromossome fitness in tests with variable mutation

There may possibly be numerous considerations that can be made about these results. A possible one is that there seem to be pairs of p_{cross} and p_{mutat} that produce better results than others (with this implementation of the GA). If, for instance, "good results" are considered to be those in which the values of the best chromosome fitness and of the population average fitness are both greater than 0.8 for the three seeds used in the tests, then the subset of tests shown in table 3.1 is obtained. In this table the tests that are considered to have the best results are highlighted (set 1 refers to tests with constant probability of mutation and set 2 refers to tests with variable probability of mutation).

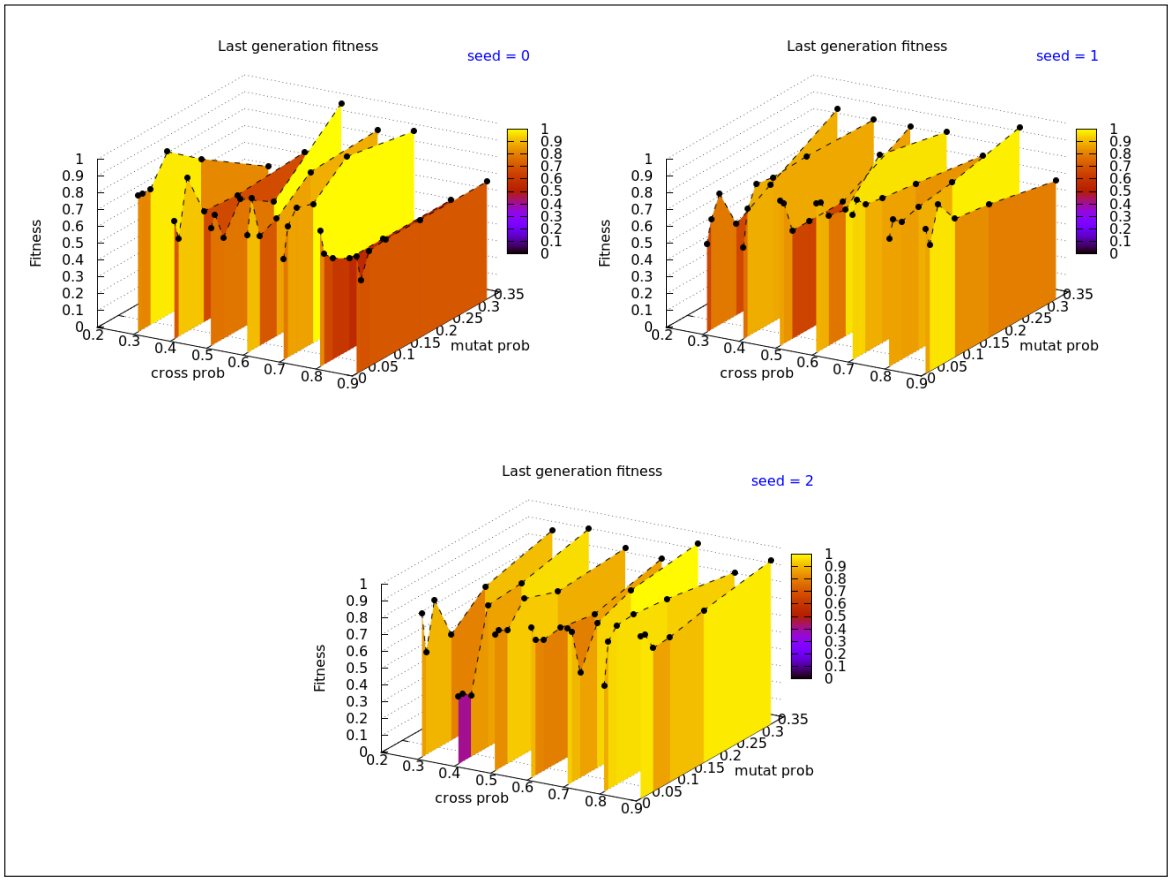


Figure 3.10: Population average fitness in tests with variable mutation

Table 3.1: Subset of selected tests

Figure in test set 1	Cross prob	Mutat prob	Figure in test set 2
	0.3	0.01	
	0.3	0.02	
	0.3	0.04	
	0.3	0.08	
	0.3	0.16	
	0.3	0.32	
	0.4	0.01	
	0.4	0.02	
	0.4	0.04	
	0.4	0.08	
	0.4	0.16	
	0.4	0.32	
	0.5	0.01	
	0.5	0.02	
	0.5	0.04	
	0.5	0.08	
Figure A.17	0.5	0.16	
Figure A.18	0.5	0.32	Figure A.60
	0.6	0.01	
	0.6	0.02	
	0.6	0.04	
	0.6	0.08	
Figure A.23	0.6	0.16	
	0.6	0.32	Figure A.66
	0.7	0.01	
	0.7	0.02	
	0.7	0.04	
Figure A.28	0.7	0.08	Figure A.70
Figure A.29	0.7	0.16	Figure A.71
	0.7	0.32	
	0.8	0.01	
	0.8	0.02	
	0.8	0.04	
	0.8	0.08	
	0.8	0.16	
	0.8	0.32	
	0.9	0.01	
	0.9	0.02	
	0.9	0.04	
	0.9	0.08	
	0.9	0.16	
	0.9	0.32	

From the analysis of that subset, some guidance can be taken for the parameterization of the GA with respect to the probabilities of crossover and mutation, namely:

- It only matters to consider a crossover probability between 0.5 and 0.7.
- If the crossover probability is 0.5, it will be necessary to use mutation probabilities equal to or greater than 0.16 if they are constant, or 0.32 if it is variable. This result suggests that for low crossover probabilities it is necessary to achieve the variability of individuals (chromosomes) through a high mutation value. And if the probability of mutation decreases over the course of evolution, then it will have to have an even higher initial value.
- If the crossover probability is 0.6 and the mutation probability is constant, then the best values for the mutation probability will be somewhere within the range $p_{mutat} \in [0.08, 0.32]$, with geometric mean at 0.16. But if the mutation probability is variable, then the initial value must be shifted to the top of that range.
- If the crossover probability is 0.7, the recommended range for the mutation probability should contain the values 0.08 and 0.16, which can be extended to 0.32 if the mutation probability is variable. The fact that this extended range (of initial mutation probability values) is greater than any other range recommended for this test appears to be one advantage of using variable mutation probability.

There were many tests with good results obtained in one or two seeds but which were not selected as "good results" because in the other seed they did not reach the thresholds mentioned above. In other words, they seemed to have too much sensitivity to the initial population, which removes robustness to the GA. In any case, these tests seem to show that the GA has a non-negligible sensitivity to the initial population (that is, to the seed of the random generator used in the implementation of the GA), which will eventually be mitigated in future tests by choosing the mutation and crossover probabilities according to the criteria described above.

As has already been mentioned, the robustness of the program that implements the GA has been an ever-present concern throughout this work, particularly due to the intense use of parallelism (and its propensity to "pop up" with parallelism-related bugs). Thus, on numerous occasions, tests were carried out whose aim was just to try to detect bugs in the program related to parallelism. For this reason, the 252 tests described in Appendix A were performed twice, with and without parallelism, comparing results that turned out to be the same, therefore increasing confidence in the version of the program developed so far.

3.2.3 Genetic algorithm in digital circuit synthesis

In order to verify the ability of the GA to generate circuit topologies, tests were performed for the synthesis of simple logic gates. In these tests the GA is expected to synthesize a circuit topology without any concerns in component sizing. This was accomplished by predefining all available components, so, in these tests, the only available components are

NMOS transistors (already sized and all equal) and PMOS transistors (also already sized and all equal). Therefore, by presetting all available components, the component sizing phase is not needed, leaving the algorithm concerned only with the topology synthesis phase.

In these tests a set of transistors is available to the GA for achievement of the objective of generating a logic gate, and each chromosome has a fixed number of genes, where each gene codifies a transistor. To simplify the task, the bulk of all transistors is pre-connected, either to V_{DD} or ground, therefore, each gene represents only the drain, gate, and source terminals of the transistor, encoding the nodes to which they are connected in the circuit by a 12-bit string. Each node is codified by 4-bit, using either a natural binary code or a Gray code (this is an option in the GA parametrization) as exemplified in Figure 3.11.

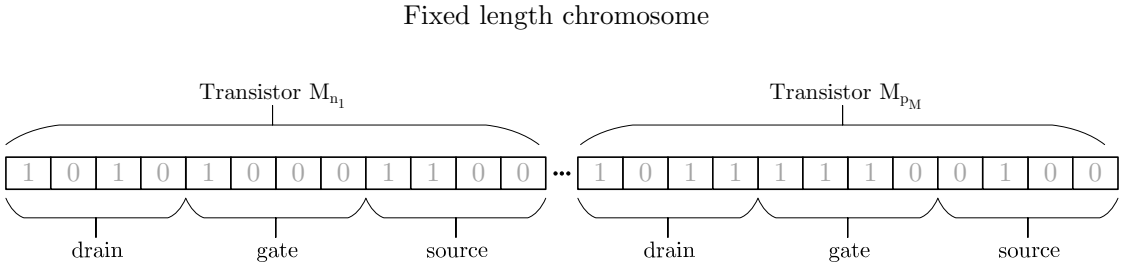


Figure 3.11: Bit string coding of transistors.

At the beginning of each run all terminals of every transistor are connected to predefined nodes (usually ground or V_{DD}, as seen in Figure 3.12) and then, gradually, the GA assigns nodes to the terminals of the transistors, thus constructing the circuit until it eventually satisfies the truth table of the intended logic gate.

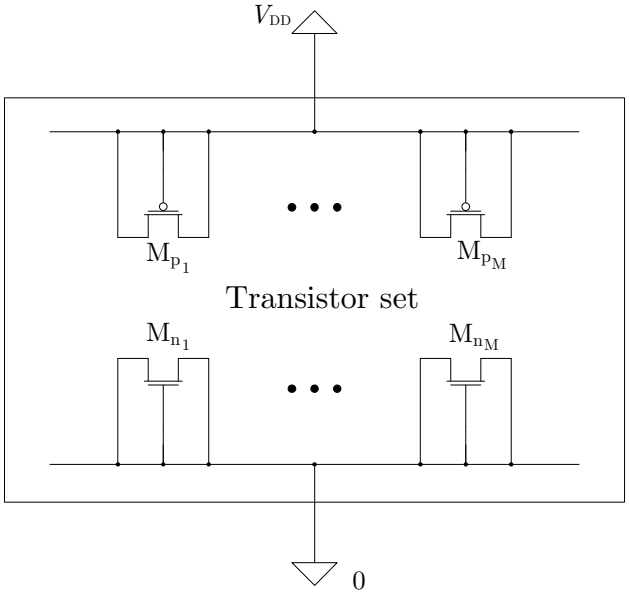


Figure 3.12: Transistor set available for the GA.

In these tests it was used elitism (with two elite chromosomes) and results from the

study presented in Section 3.2.2 were applied in the parameterization, *i.e.*, the crossover probability was constant but the mutation probability is variable (linearly decreasing in the course of evolution), both using values from that study.

3.2.3.1 Synthesis of a 2-input NAND gate

In this test the GA was tested in the synthesis of a 2-input NAND gate. The embryo circuit used is shown in Figure 3.13, where the evolvable part of the circuit is the transistor set and the rest of the components form the *context* circuit.

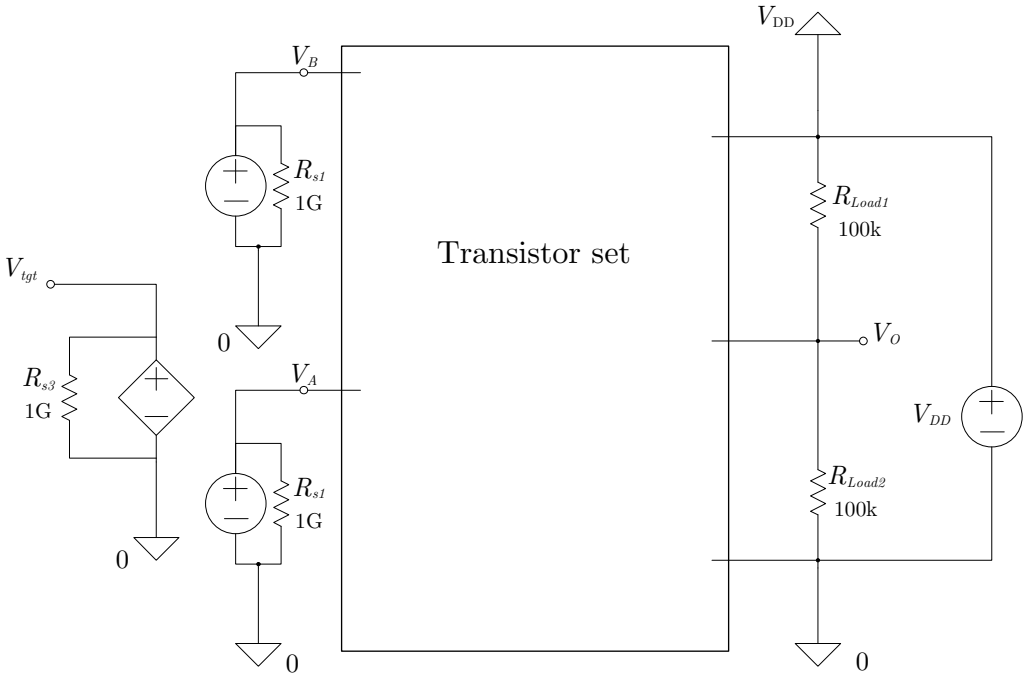


Figure 3.13: Embryo circuit for the NAND logic gate.

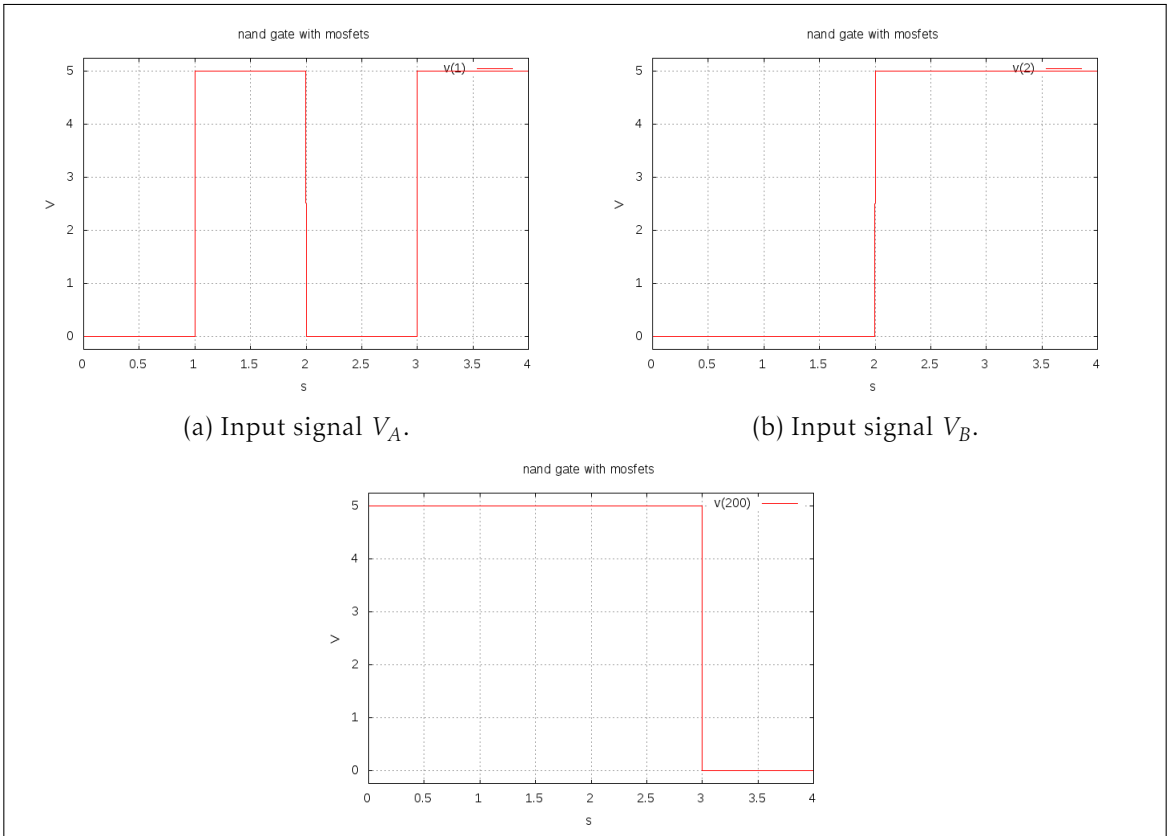
Signals V_A and V_B are the input signals to the circuit and V_{tgt} is the desired signal that should be obtained at the circuit's output node V_o, as seen in Figure 3.14. The power supply available to the evolvable circuit is $V_{DD} = 5\,\text{V}$ and the two load resistors R_{Load_1} and R_{Load_2} ensure that the generated circuit must be able to source and sink current.

Using four transistors in the embryo circuit

When four transistors are used in the embryo circuit, this embryo circuit is translated for simulation by the netlist shown in Figure 3.15. It is a netlist like this one that is submitted to NGSPICE in each iteration of the GA, in which the nodes to which the transistors terminals are connected to are updated by the GA.

In tests where four transistors were used in the evolvable transistor set (two NMOS and two PMOS), the GA produced (typically in less than 100 generations) the expected circuit, *i.e.*, the classical 2-input NAND CMOS topology depicted in Figure 3.16.

The fitness function used by the GA is given by eq. 3.9, where e_{rms} is given by eq. 3.8.



(a) Input signal V_A.

(b) Input signal V_B.

(c) Desired (target) signal.

Figure 3.14: Test signals for a NAND logic gate.

```
NAND GATE with MOSFETs
VDD 7    0    5V
Rload1   7    3    100k
Rload2   3    0    100k
* PMOS Transistors
M1 7     7    7    7    MY_PMOS W=40u L=0.8u
M2 7     7    7    7    MY_PMOS W=40u L=0.8u
* NMOS Transistors
M5 0     0    0    0    MY_NMOS W=40u L=0.8u
M6 0     0    0    0    MY_NMOS W=40u L=0.8u
* Input sources
Va   1    0    DC 0 PWL (0 0 .999999 0 1    5    1.999999 5   2    0    2.999999     0    3    5)
Rs1 1    0    1G
Vb   2    0    DC 0 PWL (0 0    1.999999 0   2    5    )
Rs2 2    0    1G
* Target ouput
Vtgt     200 0    DC 0 PWL (0 5    2.999999   5      3    0    )
Rtgt     200 0    1G
* Models
.model MY_PMOS PMOS level=1 VTO=-0.8 KP=40e-6 LAMBDA=0.1
.model MY_NMOS NMOS level=1 VTO=0.7 KP=160e-6 LAMBDA=0.1
* Analysis
.tran 100u 4
.end
```

Figure 3.15: Spice netlist for embryo circuit.

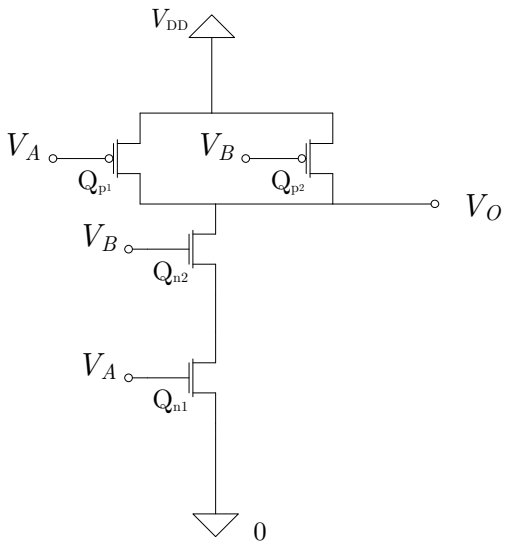


Figure 3.16: Circuit for a four transistor NAND logic gate.

$$e_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}\left(V_o(n) - V_{tgt}(n)\right)^2} \tag{3.8}$$

$$f_{fit} = \frac{1}{e^{e_{rms}}} \tag{3.9}$$

Figure 3.17 shows the average fitness of all chromosomes in the population (green line) and the fitness of the best chromosome (red line) for a complete (and successful) run of the GA that generated the circuit of Figure 3.16.

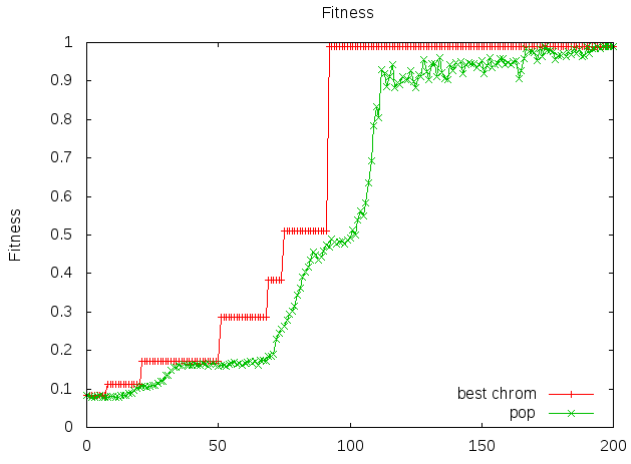


Figure 3.17: Fitness evolution in the synthesis of a four transistor NAND logic gate.

Using more than four transistors in the embryo circuit

If the set of transistors in the embryo circuit contains more than four transistors, the GA tends to produce "strange" circuits that comply with the truth table but in which there are transistors with questionable utility, such as parallel transistors and other topologies that

are either redundant or easily simplified. Note that the fitness function does not include in any way the number of transistors used (or, in this context, transistors that are no longer connected as they were in the embryo circuit), since it only considers compliance with the truth table.

Figure 3.18 shows one of these circuits, which originated from an embryo circuit with eight transistors in the transistor set (four NMOS and four PMOS). This circuit complies with

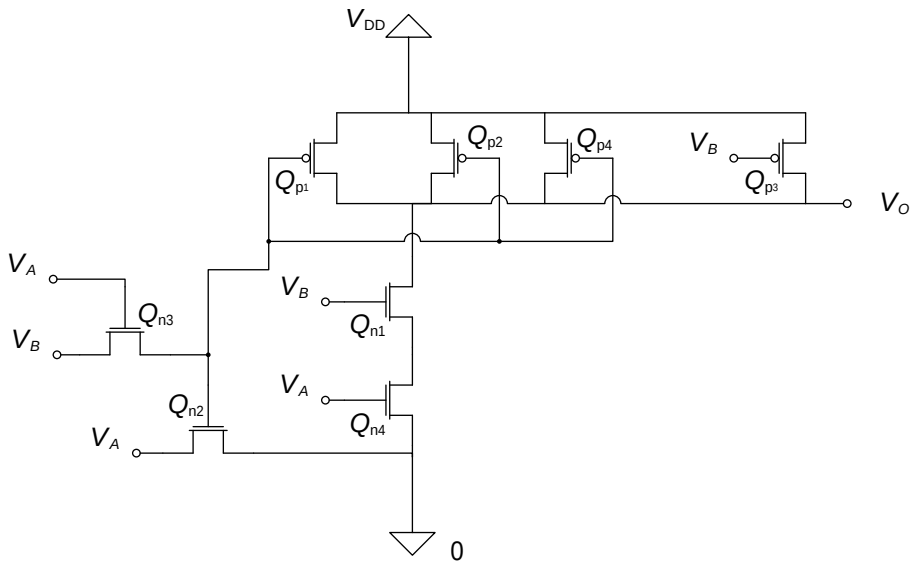


Figure 3.18: Evolved NAND circuit using 8 transistors.

the truth table of this logic gate, as shown in Figure 3.19, but uses more transistors than is necessary.

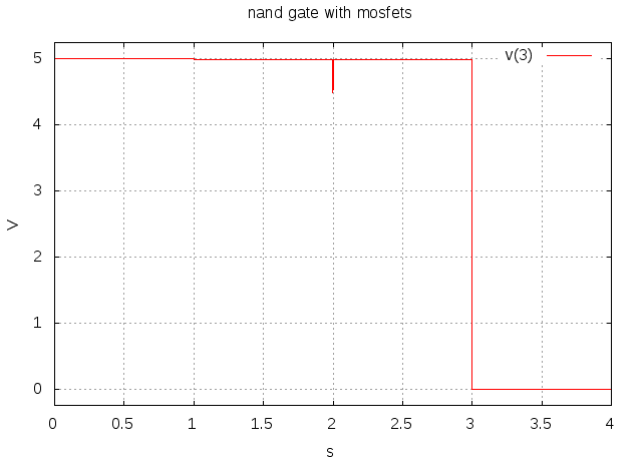


Figure 3.19: Output signal for the 8 transistors NAND circuit.

And there is also the issue, common to many other circuits synthesized to solve this problem, which is the needless consumption of current from the voltage sources that generate the input signals. In this case, this excessive consumption occurs at source V_A through Q_{n_2}, which happens when both input bits are '1', that is, when $V_A = V_B = 5\,\mathrm{V}$. This problem is

illustrated in Figure 3.20: in the fourth state of the truth table, for $t \in [3.0, 4.0[$, current consumption from source V_A reaches almost 80 mA, which is unacceptable.

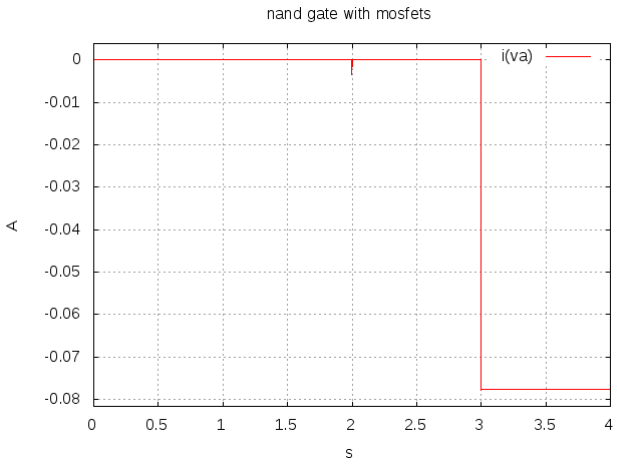


Figure 3.20: Current consumption from source V_A.

This question, of an excessive current consumption from input sources of the circuit, is similar to the question, also frequent in synthesized circuits to solve this problem, of the useless current consumption from power supply source(s).

Besides the excessive consumption that occurs at source V_A in one state of the truth table, the circuit of figure 3.18 has another issue related to the "floating" state of the node shared by the gates of transistors Q_{n2}, Q_{p1}, Q_{p2} and Q_{p4} when $V_A = 0$ (Q_{n3} is cut off). This issue was not detected by the GA because the circuit simulator "assigned" a state to that node that is the one that is required to fulfill the truth table. But in a real implementation of this topology this is a problem that could lead to undesired behavior of the circuit. A possible solution for this type of issues will be referred later in Section 3.4.1.

In order to try to reduce the number of circuits generated with the excessive current consumption issues, a new fitness function was considered that contemplates constraints regarding the currents in sources V_A, V_B and V_{DD}. This new fitness function, which was named the *global fitness function* f_{glb}, is given by eq.3.10, where all partial functions f_{V_O}, f_{I_A}, f_{I_B} and $f_{I_{DD}}$ have the form of eq. 3.11.

$$f_{glb} = f_{V_O} \times f_{I_A} \times f_{I_B} \times f_{I_{DD}} \tag{3.10}$$

$$f = 1 - e^{-weight^{-1} \times (desired/achieved)} \tag{3.11}$$

In eq. 3.10, functions f_{I_A}, f_{I_B} and $f_{I_{DD}}$ are considered constraints in spite of having the same format as function f_{V_O}, which is the fitness related to the objective function.

There are many types of constraints used in the context of GAs [Goldberg, 1989, Deb, 2001, Coello, 2002]. The expected effect of the constraints used in this problem is that they

only interfere marginally with the value of fitness for viable solutions (because they will be approximately equal to 1.0) but that they penalize unfeasible solutions as intensively as the failure to comply with the constraint (there is no need for proportionality, just monotonicity: the penalizing factor should grow with the amount of constraint violation). According to a classification suggested by Coello [Coello, 2002], this constraint functions are of the *external static* type.

The type of function depicted in 3.11 is sometimes used in the context of GAs as a minimizing fitness function [Paulino et al., 2001][Serra, 2017, pp. 74], where *desired* represents the desired value for a given variable that the GA is trying to minimize, *achieved* is the value actually achieved for that variable in a given generation, and *weight* is a factor that acts in the steepness of the function (higher values increase the steepness of the function).

Results obtained using the fitness function of eq. 3.10 showed that it prevents the synthesis of circuits suffering from the issues referred previously, *i.e.*, excessive current drawn from sources V_A, V_B and V_{DD}. But those results also showed that other issue remained unsolved: the excessive number of transistors used in circuits that would be achievable with only a reduced set of transistors (like four, in the case of the NAND logic gate). To try to solve this issue, the number of transistors used in a circuit was included in the fitness function, favoring circuits that use less transistors than others that use a greater number. The criterion used to consider that a transistor is not being used in a circuit was that it had the drain, the gate, and the source all connected to the same node (as it happens to all transistors at the beginning of a run). These transistors are removed from the netlist that is submitted to NGSPICE, which accelerates the simulations of these circuits, and are accounted for in the new fitness function, which is given by eq. 3.12.

$$f_{glb} = f_{V_O} \times f_{I_A} \times f_{I_B} \times f_{I_{DD}} \times f_{idle_{MOS}} \tag{3.12}$$

The intention of function $f_{idle_{MOS}}$ is to favor circuits that use less transistors, being able to take advantage of its knowledge of the number of transistors available in the transistor set. Several functions were tested for this purpose and the best results were obtained with function $f_{idle_{MOS}}$ depicted in eq. 3.13.

$$f_{idle_{MOS}} = (1 - f_0)\frac{n_{idle_{MOS}}}{N_{MOS}} + f_0 \tag{3.13}$$

In eq. 3.13, N_{MOS} is the total number of transistors available in the transistor set, $n_{idle_{MOS}}$ is the actual number of transistors considered *idle*, *i.e.*, that have all terminals connected to the same node, and f_0 is the minimum merit value allowed for this function. An example of these parameters used in tests trying to generate a NAND gate from a transistor set with four NMOS and four PMOS transistors is shown in eq. 3.14, where $N_{MOS} = 8$ and $f_0 = 0.9$.

$$f_{idle_{MOS}} = 0.0125 \times n_{idle_{MOS}} + 0.9 \tag{3.14}$$

Unfortunately, the use of $f_{idle_{MOS}}$ in the fitness function does not always have the expected result and often prevents the GA from converging to a useful solution. The success of the GA seems to be very sensitive to the value of f_0 and it was not possible to achieve a good balance between the intended effect of $f_{idle_{MOS}}$ and the rate of success of the GA. But, instead of rejecting the use of $f_{idle_{MOS}}$, it seems preferable to gradually introduce this function into the fitness function, leaving, for example, the GA to evolve without $f_{idle_{MOS}}$ during the first 100 or 200 iterations and only then introduce the contribution of $f_{idle_{MOS}}$. This procedure was applied to the embryo circuit that generated the circuit of Figure 3.18 and in several (more successful) cases, the generated circuit resulted in the circuit shown in Figure 3.16, that is, the GA discarded four transistors (from the set of eight) and achieved the desired functionality with only four transistors.

3.2.3.2 Synthesis of other logic circuits

In addition to the NAND gate already presented, attempts were made to synthesize other logic circuits, such as a NOR gate, an AND gate, an XOR gate, and a half-adder. Results regarding the synthesis of a NOR gate are very similar to results obtained with the synthesis of the NAND gate and will not be shown here. But results about other topologies were not as expected and were quite disappointing.

Synthesis of a AND gate

In addition to NAND and NOR gates, an attempt was also made to synthesize an AND gate, which proved to be a much more complex task for the GA. After numerous runs with different parameterization and different seeds (for the pseudo-random generator), the best circuit synthesized by the GA is shown in figure3.21. This circuit was obtained by the evolution of an embryo circuit with 48 transistors (24 NMOS and 24 PMOS) but function $f_{idle_{MOS}}$ was not used since it prevented the convergence of the GA in all tests performed. And for smaller transistor sets the GA was never able to generate a circuit that fulfills the intended truth table. Although this circuit fulfills the truth table of an AND gate, this is clearly not a satisfactory result.

Synthesis of a XOR gate and a half-adder

Using the knowledge gathered with previous tests, further attempts were made in the synthesis of a XOR gate and an half-adder circuit. In spite of all the effort dedicated to this task, it was never possible to obtain a circuit that fulfills the truth table of such functions (Table 3.2). In all tests the GA managed, at most, to comply with three of the four states of the truth table.

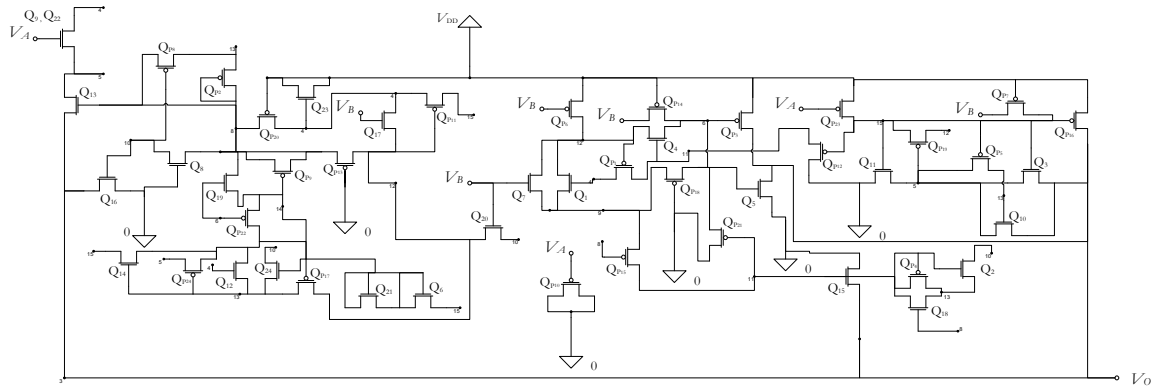


Figure 3.21: Generated circuit for the AND logic gate.

Table 3.2: Truth table for a XOR and an Half-adder

Inputs		Outputs		
		XOR	Half-adder	
			carry	sum
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	0	1	0

3.2.3.3 Limitations

Results obtained with these tests show that the GA is able to synthesize simple logic circuits, like NAND and NOR, and even though they are sometimes accomplished with rather unusual topologies, other times the well-known classic topology of such gates is produced. Other gates, like AND and OR are much harder to synthesize by the GA and the very few circuits generated that fulfilled the truth table had much more transistors than the classic topology of such gates. But the GA failed to synthesize other circuits, like a XOR or a half-adder.

Considering the knowledge gained after numerous tests trying to synthesize logic gates, one idea that stands out is that the GA has to have access to a number of transistors that must be much higher than what is actually necessary to carry out the final circuit, so that there is a useful capacity for exploration in the early stages of evolution. But then there will also have to exist a process for the GA to discard all transistors that do not contribute to the objective function, which will otherwise stay "hanging around" in the final circuit.

Although it was possible to accomplish the generation of simple circuits, the GA clearly suffers from lack of scalability to higher complexity topologies. Without entire certainty, it was hypothesized that this lack of scalability is essentially related to the coding scheme used by the GA. Therefore, it was necessary to try another codification technique, and VLCs were tested.

3.3 Variable length chromosomes

In this work VLCs were tested as an encoding scheme, in which a chromosome is a circuit descriptor and each gene is a component descriptor [Campilho-Gomes et al., 2020]. Each chromosome has a variable number of genes, and this number may either increase or decrease during execution of the GA. In this encoding scheme each gene is comprised of three fields, as depicted in Figure 3.22. The first field identifies the component type and

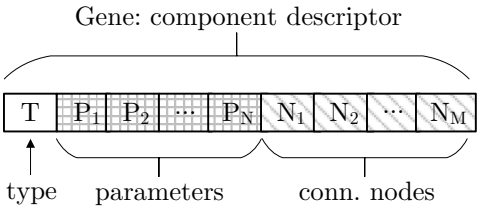


Figure 3.22: Component encoding using VLCs.

is typically implemented as an integer variable. The second field is a set of parameters needed to describe the characteristics of the component, which are typically implemented as real variables. And the third field is a set of nodes indicating where the component terminals are connected (implemented as positive integer variables). An example of this circuit encoding scheme is shown in Figure 3.23, in which a chromosome with three genes encodes a simple circuit with three components: a MOS transistor (p channel), a bipolar transistor (npn) and a resistor, whose parameters and device connections are clarified in table 3.3.

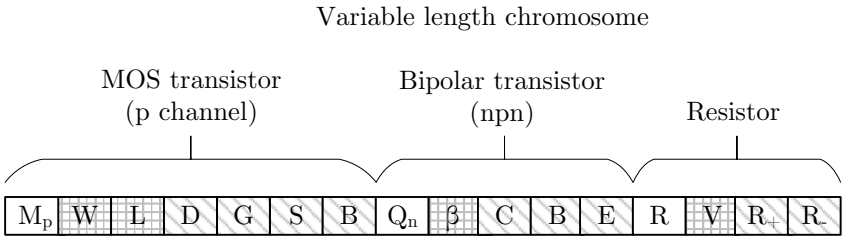


Figure 3.23: Circuit encoding using VLCs.

3.3.1 Crossover and Mutation

The use of VLCs as an encoding scheme to describe circuits led to some adaptions on both crossover and mutation operators. Although the purpose of these operators in the GA that uses VLCs is equivalent to their purpose in GA-c, the original operators proposed by Holland in the GA-c need to be modified so that they work with real variables and VLCs.

3.3.1.1 Modified Crossover

In this work, a gene boundary crossover operator is used that implements equal crossover and inside crossover operations [Deif and Gadallah, 2014] as illustrated in Figure 3.24. Using this operator to cross two chromosomes to generate two offspring, the shortest

Table 3.3: Example of parameters and connections

Type	Allele function	Symbol	Description
MOS transistor	Parameters	W	channel width
		L	channel length
	Connections	D	drain terminal
		G	gate terminal
		S	source terminal
		B	bulk terminal
Bipolar transistor	Parameters	β	current gain
	Connections	C	collector terminal
		B	base terminal
		E	emitter terminal
Resistor	Parameters	V	value (in Ω)
	Connections	R_+	terminal 1
		R_-	terminal 2

chromosome length never decreases and the longest length never increases, so the operator does not change neither *maxlength* (the length of the biggest chromosome in the population) nor *minlength* (the length of the smallest chromosome in the population): the ability to change either *maxlength* or *minlength* is confined to the mutation operator.

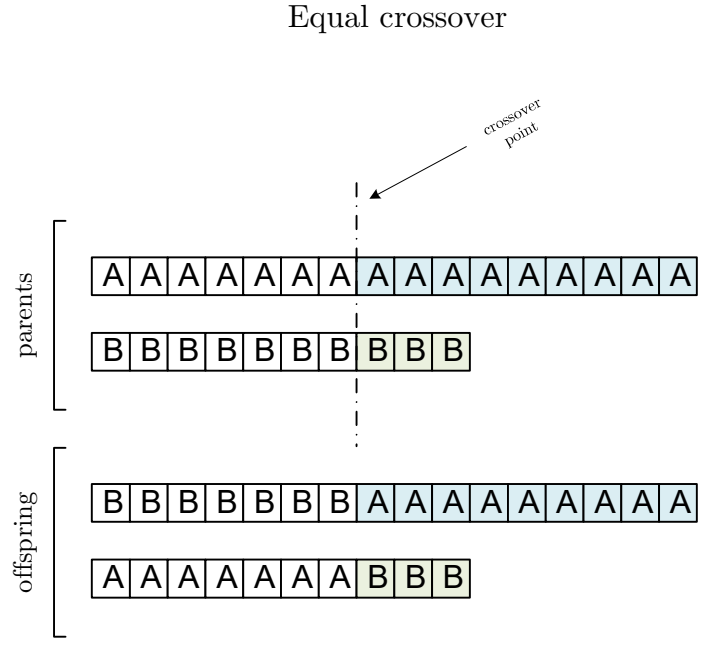


Figure 3.24: Crossover operator for variable length chromosomes.

3.3.1.2 Modified Mutation

The mutation operator used can, potentially, alter any allele of a gene. This means that it can change all characteristics (Figure 3.22) of any gene in the circuit, namely its type,

any of its parameters, and the node to which any terminal is connected. Although any of those mutations is hypothetically possible, the mutations that are actually allowed for a given run of the GA can be specified, and some of them are often not allowed because they would result in an incoherent component description. For instance, changing the type of a gene from a bipolar transistor to a resistor (by just mutating the allele that describes the component type) would leave the component in an incoherent state because (at least) the number of terminals would not match what is expected (two terminals for a resistor instead of three inherited from a bipolar transistor). Another approach would be to allow the mutation but penalize the fitness of any chromosome with incoherent components.

3.3.1.3 Other features of the mutation operator

The mutation operator also implements *insertion*, *replication* and *deletion* operations [Hutt and Warwick, 2007], which are responsible for changing *maxlength* and *minlength* in the population by: 1) inserting a small piece of new genetic material in a chromosome; 2) replicating a small piece of genetic material within a chromosome; 3) deleting a small piece of genetic material from a chromosome.

Another feature of the mutation operator is the ability of rejecting a mutation if it worsens the chromosome fitness. The probability p_r of rejecting a mutated chromosome is not constant, rather it is adapted as the population evolves. The probability p_a of acceptance ($pa = 1 - p_r$) starts high and decreases over time, similarly to a simulated annealing process [Barros et al., 2010], allowing broader regions of space to be searched in early generations, but later narrowing this search to small regions around the solution(s) already obtained (local exploration). However, it is difficult to determine the rate at which the probability of acceptance should decrease over time (because it is difficult to determine the rate at which the temperature should decrease in a simulated annealing process to obtain an optimal solution [Nourani and Andresen, 1998]).

Instead of decreasing the probability of acceptance over time (the standard procedure in classic simulated annealing), this work uses a heuristic that adapts this probability according to a function f_Q that measures the quality of the solutions obtained so far (and converts this measure of quality into a probability). This measure is obtained by the average fitness of the N best chromosomes in the actual population, smoothed by a 2nd order discrete low-pass filter (LPF). Function f_Q is bounded [0..1], where 0 represents the worst possible case and 1 represents the best possible case. The LPF is implemented by cascading two first-order IIR low-pass filters described by 3.15, and function f_Q is described by 3.17, where $fit(P_i)$ is the fitness of the population in iteration i given by 3.16, where N is the number of best chromosomes being considered ($N = 1$ if only the best chromosome is considered, $N = 2$ if only the best and second-best chromosomes are considered, and so on), and $fit(C_j)$ is the fitness of chromosome j.

$$y[n] = (1 - \alpha)x[n] + \alpha y[n - 1] \tag{3.15}$$

$$fit(P_i) = \frac{1}{N}\sum_{j=1}^{N} fit(C_j) \tag{3.16}$$

$$f_Q(P_i) = LPF\langle fit(P_i)\rangle \tag{3.17}$$

To map a fitness value (obtained by f_Q) to a probability value (the probability p_r of rejection), we use a simple nonlinear heuristic of an exponential nature, function f_{map}, described by 3.18 and depicted in Figure 3.25.

$$f_{map}(x) = \frac{(y_M - y_m)(e^{kx} - 1)}{e^k - 1} + y_m \tag{3.18}$$

This function $f_{map}(x)$ uses parameter k to adjust the steepness of its elbow shape, where $x \in [0\,..\,1]$, and $y \in [y_m\,..\,y_M]$, which, in this case, are set to $y_m = 0$ and $y_M = 1$. In numerous successful tests for digital and analog circuit generation, the parameter k has been used in the range of 1 to 4; nevertheless, further adjustment may be required in other contexts.

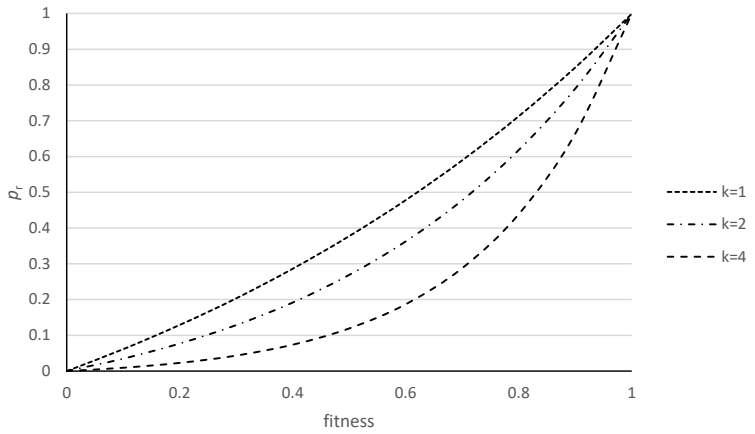


Figure 3.25: Function f_Q used to map fitness to probability p_r.

This method of measuring the quality of the solutions performed quite satisfactorily in all the tests conducted, both for the synthesis of digital and analog circuits, showing that it is a robustness enhancer of the adaptive process, since it reduces its sensitivity to the convergence speed of the GA (which is known to be highly irregular and dependent on numerous factors, such as the initial solution and the general GA parameterization). The overall process can be summarized by the pseudo-code shown in Algorithm 1 on the next page.

A relevant advantage of this technique is that it reduces the sensitivity of the GA to the probability parameters used by the mutation operator (the plural "parameters" is used here because, when using VLCs, there are often several different probability parameters involved in the mutation operator).

Algorithm 1 Mutation of a chromossome

```
function chromMutation
    if it is time to mutate this chrom then
        Make a copy of this chrom
        Mutate this chrom
        Evaluate mutated chrom
        if mutated chrom has better fitness then
            Keep mutated chrom in the population
        else
            Calculate f_Q using population info
            Calculate f_map
            Calculate probability p_a of acceptance
            if p_a is high enough then
                Keep mutated chrom in the population
            else
                Keep unmutated chrom in the population
            end if
        end if
    end if
end function
```

3.4 Additional techniques

In the course of this work some additional techniques that are specific for circuit related issues developed by GAs were implemented and used in some tests described in this document. These techniques, entitled "Circuit test bed", "Parallel association", and "Multiphase parametrization", aim to improve the circuits generated by the GA and were developed as certain situations were found in the course of this work. The "Circuit test bed" is a pseudo-fixture that resembles a bed-of-nails test fixture, the "Parallel association" deals with parallel association of components and "Multiphase parametrization" segments the evolution of the GA in several autonomously parameterized phases.

3.4.1 Circuit test bed

It is very common to have components with floating nodes in circuits generated by a GA. These circuits are normally rejected by SPICE-like simulators because analysis methods used by SPICE require that every node in a circuit have a DC path to ground. One technique to deal with this problem that is often referred in the literature [Koza et al., 1997c,Sapargaliyev and Kalganova, 2006a,Sapargaliyev and Kalganova, 2006b,Sapargaliyev and Kalganova, 2006c, Lohn et al., 2000] is to connect each of those nodes with a auxiliary resistor to some predefined node, usually to ground using 1 GΩ resistors. This procedure of connecting every floating node to ground by an auxiliary resistor was included in the current implementation of the GA as an additional technique that can be enabled whenever the GA kernel is being used to synthesize circuits. The intention of this procedure is to allow such circuits to be simulated, providing genetic material for younger generations

that the GA will hopefully evolve to a solution that fulfills the circuit's objective. It is expected that those auxiliary resistors end up being removed by other techniques that eliminate useless components in the circuits. In fact this happens very often but not always, so sometimes the GA generates circuits that still include few of those auxiliary resistors.

Another problem, related to this one and often found in circuits generated by GAs, is the existence of MOS transistors whose gate is left virtually floating in some circuit states. For instance, if the gate of a particular MOS transistor T_1 of a logic gate is connected only to the drains of two other MOS transistors T_2 and T_3 and there is one state that cuts off both transistors T_2 and T_3, then a human circuit designer would probably consider the gate of T_1 being floating, which would almost certainly be considered undesirable. But SPICE does not consider this gate as a floating node because in fact there is a DC path to ground, and therefore the circuit is not rejected for simulation. Even though transistor T_1 would be considered to be in an undefined state in a real world circuit, in SPICE simulation it may eventually assume a state that is the desired one considering the intended circuit operation, which can lead to a misclassification of the circuit as a "good" circuit. However, this classification will eventually be revoked in a later stage of assessment since the circuit will almost certainly not withstand a test with some noise injection because that state will eventually change by effect of the noise signal.

To illustrate this problem with a circuit generated by the GA in a run in which this technique was not used, consider the circuit shown in Figure 3.26 (which was already presented in Section 2.2.1.2 and is reprinted here for convenience).

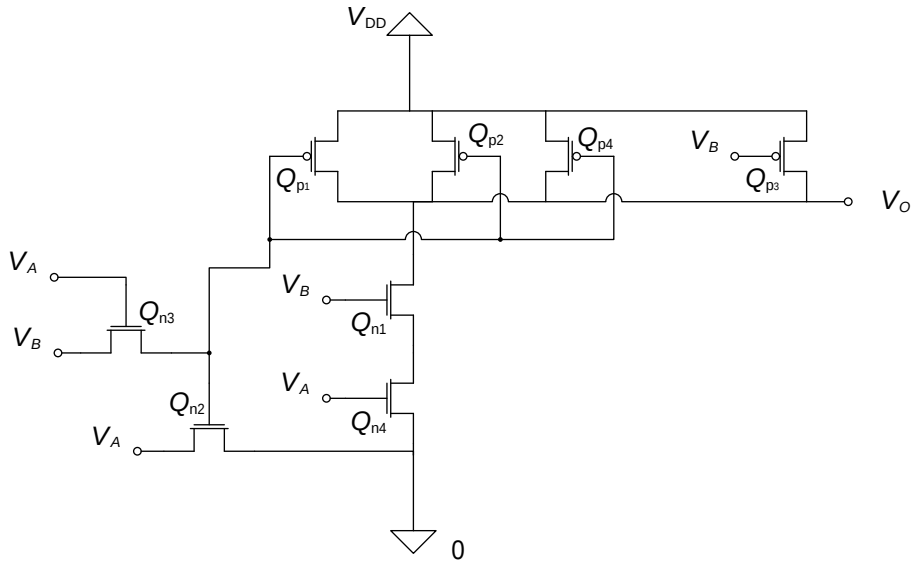
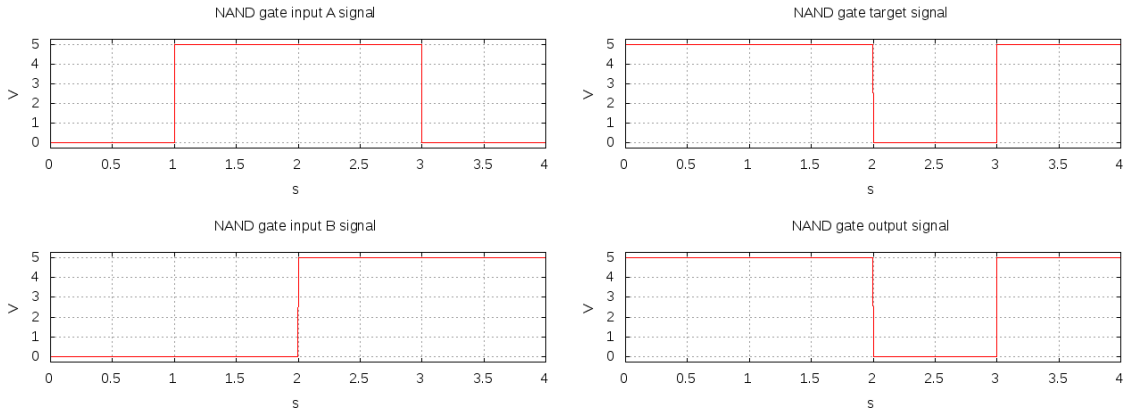


Figure 3.26: Evolved NAND circuit.

This circuit was generated by the GA as a possible solution for a two-input NAND gate, and in fact this circuit fulfills the truth table for a NAND gate as shown in Figure 3.27.

However, the node shared by the gates of transistors Q_{n2}, Q_{p1}, Q_{p2} and Q_{p4} has the problem referred above: it enters a state that is difficult to predict when $V_A = 0$ (Q_{n3} is cut off).



(a) Input signals (b) Target and output signals

Figure 3.27: Input, target (desired), and output signals.

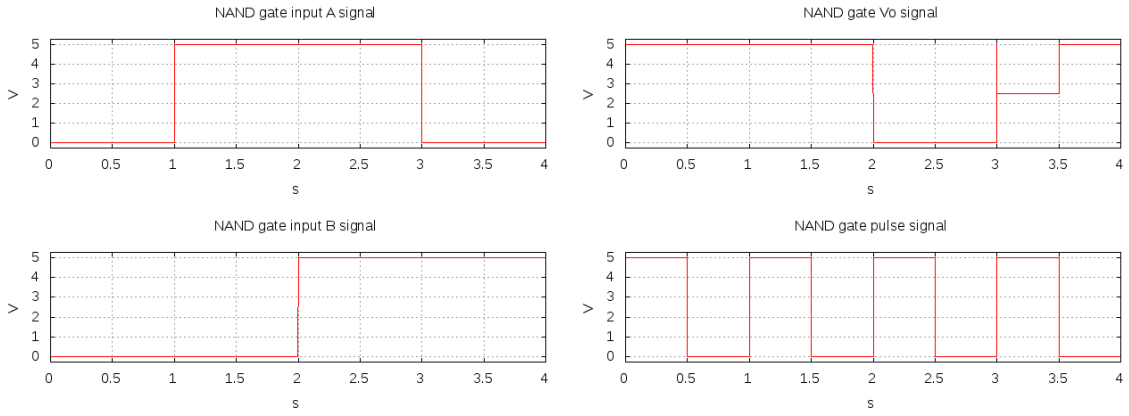
This would be considered an undefined state by a human designer as the node would be considered a floating node.

To avoid the existence of such floating transistor gates, connecting every transistor gate to ground with a high value resistor was tested. This was an expansion of the procedure previously described, which was applied before to floating nodes and now was also applied to transistor gates. This would hopefully define the gate state as logic '0', removing some unpredictability to the circuit behavior but would add undesired resistors to the circuit. And anyway, as far as good noise rejection is concerned, it would never be acceptable to leave a transistor gate connected to an equivalent impedance of 1 GΩ in some states. Therefore, this methodology would not solve the underlying problem, which is to ensure that all gates "see" a low impedance in all logic states.

To try to solve this issue a methodology close to this one was tested: instead of connecting every transistor gate to ground with an auxiliary high valued resistor, this resistor was connected between each gate and an auxiliary voltage source generating a rectangular pulse (that can be seen as a pseudo noise source with characteristics adapted to this issue). This pulse changes its state between logic 0 and 1 voltages and it has a frequency that is twice the frequency of change of state imposed by the input signals. This means that for each state in the truth table of a logic circuit that pulse signal has both logic values 0 and 1 (with a duty cycle of 50%). Should any transistor gate stay in a high impedance situation in some state and the effect of that pulse should force that transistor to commute "inside" that state, eventually revealing an undesired output signal.

Applying this technique to the circuit of Figure 3.26 yields the signals shown in Figure 3.28 where the signal V_o no longer fulfills the truth table because for half of the duration of the last state the output is not the desired one.

In this last state, where $V_A = 0\,\text{V}$, $V_B = 5\,\text{V}$ and V_o should be 5 V, there is a period $t_{fault} \in]3.0, 3.5[s$ during which $V_o = 2.5\,\text{V}$. During this period t_{fault} all transistors in the circuit are



(a) Input signals (b) Target and pseudo noise

Figure 3.28: Signals used in NAND test with test bed.

cut off, leaving the output in a state that is commonly known as tri-state. In this state V_o results only from the resistive voltage divider created by the two load resistors (pull-up and pull-down resistors of 100 kΩ not shown in Figure 3.26). This unwanted value of V_o penalizes the fitness of this circuit (because there is a mismatch between the desired output signal and the actual output signal) which may be enough for the GA to "choose" another circuit over this one.

The auxiliary resistors that connect each transistor gate to the pulse voltage source are not included in the circuit's chromosome, instead they are used as a *test bed* for the circuit, hence they are included in the netlist submitted for circuit simulation (the name *test bed* was used because this technique resembles a bed-of-nails tester, which is a traditional electronic test fixture used to test PCBs and assembled circuits [Wright et al., 2003, Mysore et al., 2006]).

This technique does not detect all possible situations of floating gates but solves some of them and it was successfully used in several tests of simple logic gates like NAND, NOR, AND, OR and XOR.

3.4.2 Parallel association of components

It is quite often to find components connected in parallel in circuits generated by the GA. Sometimes these components only appear in intermediate circuits that the GA generates during evolution, *i.e.*, they are only seen in intermediate generations and do not endure till the end of the run, so they do not appear in the final circuit. However, other times the final circuit produced by the GA has several sets of parallel components that could be easily associated in a single component each.

The occurrence of such parallel components may be diminished by using a penalizing factor in the fitness function that accounts for the total number of components in the circuit. Nevertheless, even doing so, it seems to be a "hard task" to the GA to get rid of all parallel

components, either taking an unreasonable number of iterations or failing in associating them all.

So, a parallel detector and associator was developed and included in the GA. The algorithm of this detector can find and associate parallel MOS transistors and passive components like resistors, capacitors, and inductors. It is normally used before the *selection* stage of the algorithm and the periodicity of usage in a test is left to a per test configuration. In order to maintain genetic diversity, it is usual to keep the circuit that has the parallel components and to add a new circuit (to the population) resulting from the application of the detector.

3.4.3 Multiphase parametrization

Following the use of variable weights, there was an interest in the potential that could come from segmenting the evolution of the GA in sections (sets of iterations) that allowed different parametrization in each, namely with discontinuous changes of some parameters between sections. These sections, entitled "phases", in which one component of the fitness function is given more importance than others, would allow for some steering of the GA directing its search and optimization effort towards a specific optimization target, like the number of components or the total area used in MOS transistors. The criterion for changing the phase can be phase stagnation or reaching a predefined threshold (given by some metric, the most immediate example being the fitness function reaching a predefined threshold).

This process can be illustrated by the diagram in Figure 3.29, which shows a three-phase example that was actually used in several tests. Each phase has its own set of parameters, that are adjusted to adjust the intended steering for each phase. Typically, two adjacent phases share most of the parameters in each parameter set, and only a few are changed (eventually only one is changed).

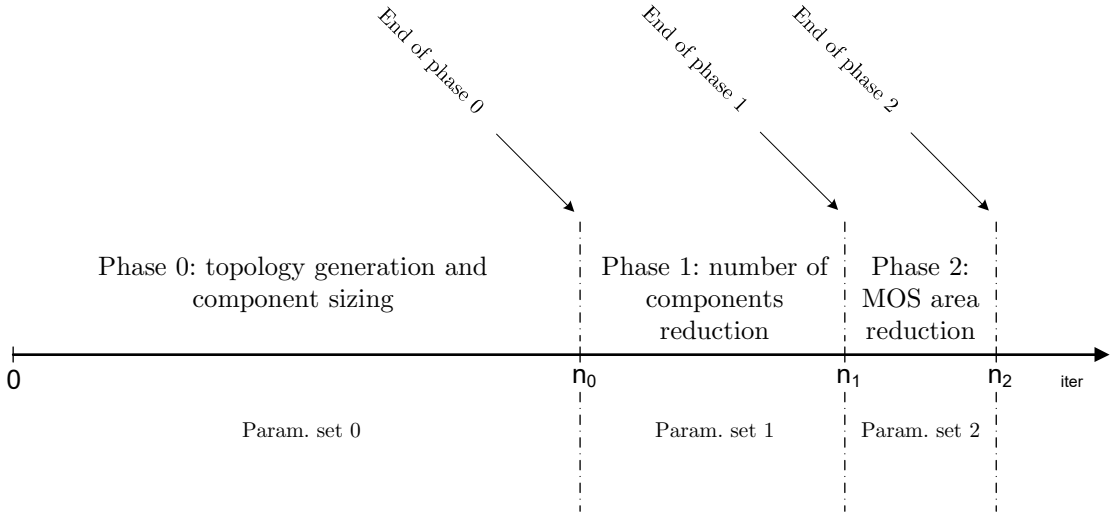


Figure 3.29: Multiphase diagram: example for 3 phases.

In this example, phase 0 uses a set of parameters (Param. set 0) that "tunes" the GA to

search for a circuit topology that satisfies some goal without being overly penalized by either the total number of components in the circuit or by the total area used in MOS transistors. Then, when a given fitness threshold is reached (or some other stopping criterion), phase 0 is terminated and another parameter set (Param. set 1) is used that has some extra penalization for the total number of components used in the circuit. When this phase stagnates it is terminated and a third parameter set (Param. set 2) is now used that shifts the optimization effort of the GA by adding extra penalization for the total area used in MOS transistors. This process is described in Figure 3.30.

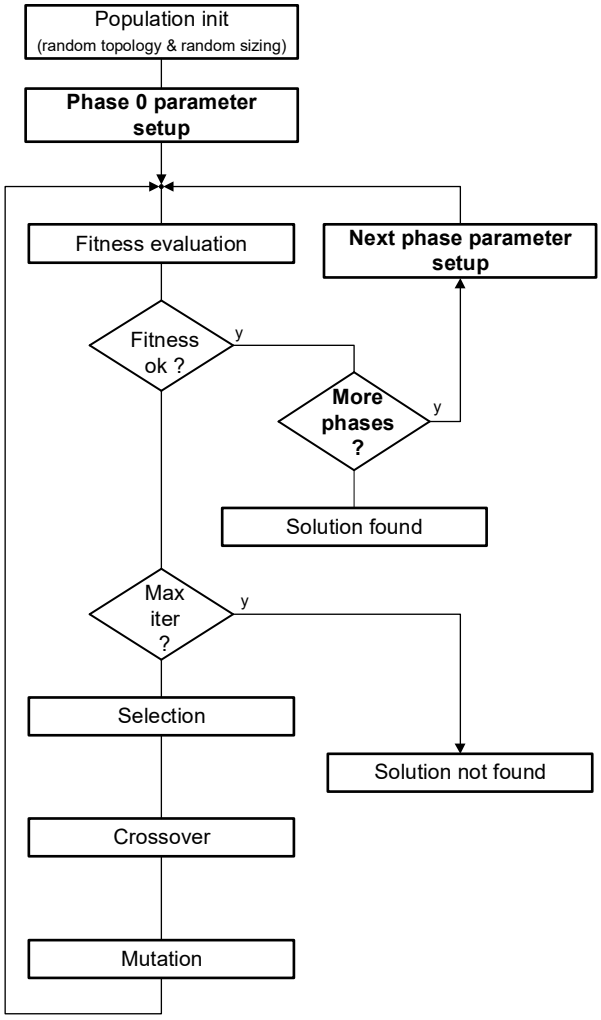


Figure 3.30: Diagram of the GA with multiphase parametrization.

This technique can be used with any number of phases and with other partial goals other than those mentioned in the previous example. According to the knowledge gathered from experimental results of several tests, this multiphase parametrization technique, complemented by the use of variable weights along evolution, accomplished better results in all the tests in which a comparison was carried out without using it. However, it is not a technique of universal use for all types of problems that are intended to be solved by a GA. This technique can be useful when there is already some knowledge about how the

GA behaves in solving a given problem, and then, in some cases, as in the example given before, this technique may prove to be appropriate.

CHAPTER 4

Experimental Results

This chapter contains experimental results obtained using the techniques introduced in the previous chapter. Digital and analog circuits synthesized automatically by the genetic algorithm are presented and discussed.

In this work there was a first attempt to use a codification technique for circuits that applies the classic bit strings used by GA-c to represent a circuit. In this representation there is a fixed number of components available to the GA and the evolution takes place by changing the nodes to which each component terminal is connected to. The circuit of Figure 4.1, which was already presented in Section 3.2.3.1 and is reprinted here for convenience, is an example of a circtui that was generated automatically by the GA using this codification technique. This particular circuit was produced in a run in which only 4 transistors were available to the GA for circuit synthesis.

Although it was possible to accomplish the generation of simple circuits with FLCs, this codification scheme suffers from lack of scalability to higher complexity topologies. Therefore, as referred in the previous chapter, it was necessary to try another codification technique, and VLCs were tested.

One of the first circuits to be successfully generated using VLCs, which had never been successfully generated while using FLCs, was the half-adder. Despite all the effort and CPU time spent on numerous attempts to generate this circuit, it has never been possible to use FLCs to obtain a circuit that at least satisfies the corresponding truth table. However, using VLCs, it was possible to generate the circuit of Figure 4.2, which, without being optimized

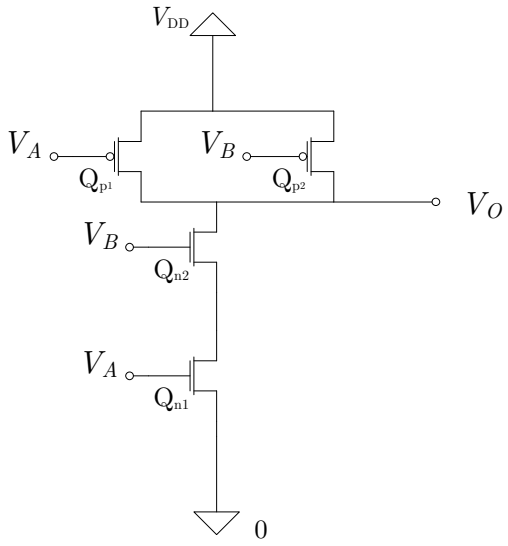


Figure 4.1: Circuit for a four transistor NAND logic gate.

in any way (namely in the number of components), fulfills the intended truth table.

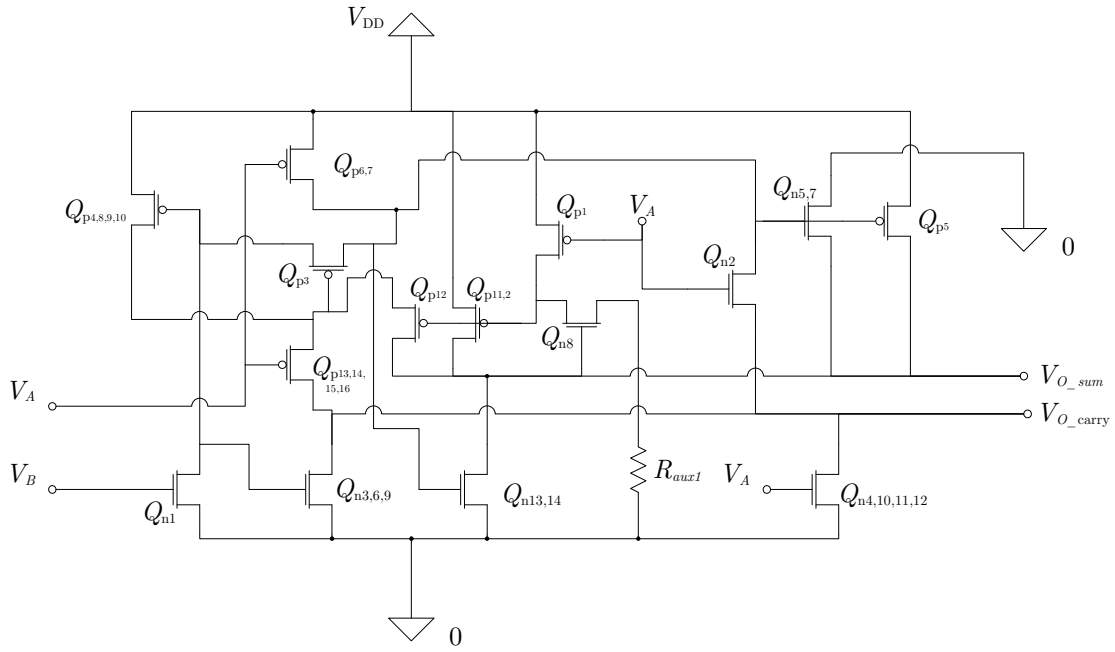


Figure 4.2: Circuit for a two-input half-adder logic gate.

The study of the previous circuit was not deepened, namely regarding the search for optimized solutions, but obtaining that circuit opened the way to tests on other types of circuits that had not yet been successful, such as analog linear amplifiers.

4.1 Synthesis of an amplifier using BJT transistors

Using VLCs with the proposed codification scheme (Section 3.3), and using a SPICE-based circuit simulator for fitness evaluation, the GA was able to generate circuits of higher

complexity, such as the half-adder shown in Figure 4.2 and the 20 dB DC amplifier (DC-coupled amplifier) presented in the next section. Fitness evaluation is performed by a SPICE-based circuit simulator, NGSPICE [SourceForge, 2019], in a simulator-in-the-loop paradigm, and all candidate circuits are submitted to simulation, i.e., there's no prediction technique [Domingues et al., 2022, Hakhamaneshi et al., 2019] (often referred to as an "oracle") that filters out candidates with poor convergence probability (or with any other acceptance/rejection metric).

4.1.1 A 20dB DC Amplifier

One test performed on the GA was the synthesis of a DC amplifier with gain 10 using bipolar transistors. In this test the GA could use resistors and either NPN or PNP transistors to synthesize the circuit, and the embryo circuit of Figure 4.3 was used as a starting point.

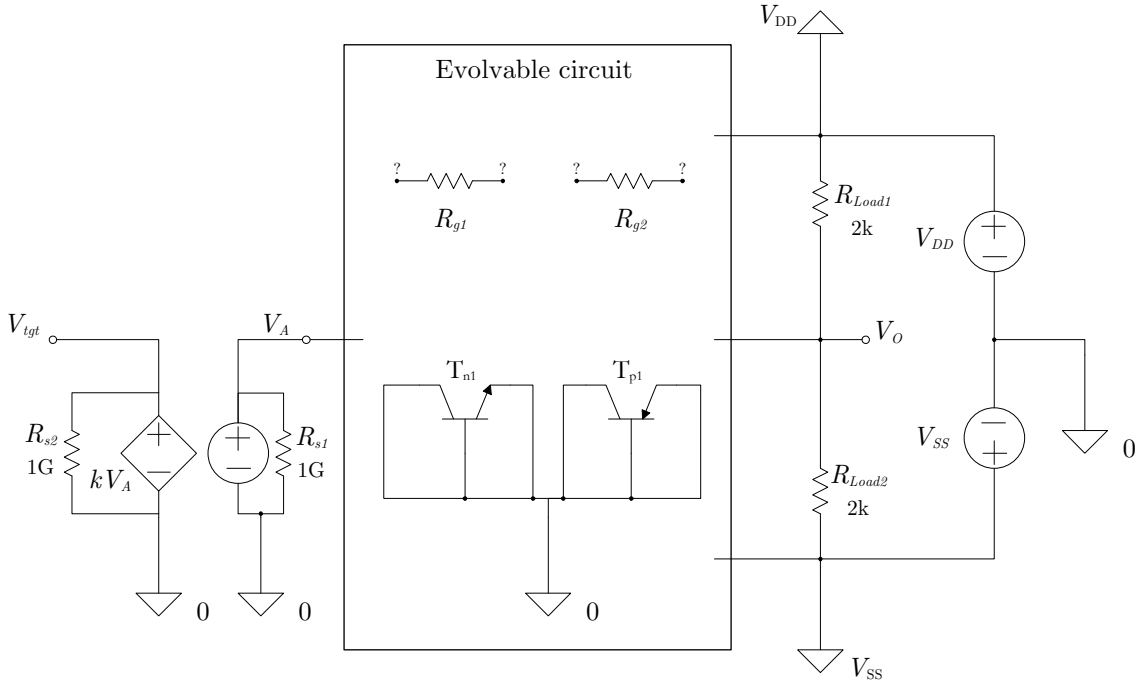


Figure 4.3: Embryo circuit for a DC amplifier with gain 10 using bipolar transistors.

4.1.1.1 The embryo circuit

In the embryo circuit, only the portion enclosed in the box titled "Evolvable circuit" is allowed to be evolved by the GA; the rest of the circuit is the *context* circuit. Signal V_A is the input signal of the amplifier and signal V_{tgt} is the desired (or target) signal that should be obtained at the amplifier's output node V_o. Some parameters of this circuit are shown in Table 4.1, while some GA parameters are shown in Table 4.2.

The netlist used to describe this embryo circuit, which is one of the input files to the program that implements the GA, is shown in Figure 4.4. This netlist is later complemented by the GA with changes (to the circuit inside the box) that may result from insertion, removal

```
DC amplifier 20dB
*
VDD          7        0        2.5V
Rvdd         7        0        1G
*
VSS          8        0        -2.5V
Rvss         8        0        1G
*
Rload1       7        3        2k
Rload2       3        8        2k
*
Va           1        0        DC  0   SIN (0 0.01 2 0)
Rs1          1        0        1G
*
Etgt         200      0        1       0       10.0
Rtgt         200      0        1G
*
.MODEL       MY_NPN   NPN      BF=100
.MODEL       MY_PNP   PNP      BF=100
*
.TRAN        100u     1
.END
```

Figure 4.4: Spice netlist for embryo circuit.

or modification of components. It is the resulting netlist that is then fed to NGSPICE for circuit simulation.

Table 4.1: Circuit parameters

Positive power supply	V_{DD}	2.5 V
Negative power supply	V_{SS}	-2.5 V
Target signal source gain	k	10
Input signal	$v_A(t) = \sin(\omega t)$	$A = 10$ mV , $\omega = 2\pi \times 2$ Hz
Transistors current gain	β_{npn}, β_{pnp}	100
Resistors in evolvable circ	$R_{g1}, R_{g2}..R_{gN}$	10 Ω to 1 MΩ
Max n of nodes in evolvable circ	$node_{max}$	10 (excluding ground)

In this embryo circuit, four components were used, two having a predefined connectivity (the transistors, which in this test were chosen to be initially connected to ground) and the other two having random connectivity (the resistors). In addition, in this test the transistors do not have any parameters configurable by the GA, but the resistors have their value determined by the GA during the entire evolution of the circuit. It is worth mentioning that, after carrying out several tests similar to this one using different initializations, it seems that the final result obtained by the GA is not significantly influenced by characteristics such as initial connectivity and initial value of the components.

With the embryo circuit and the parametrization presented, the circuit of Figure 4.5 was obtained when the fitness function (described below) reached a stopping threshold of 0.95, which occurred at iteration 1222. This circuit has many redundant or even useless components, but it accomplishes the main objective of this problem, since it produces an output signal similar to the desired one (*i.e.*, the error of "similarity" is within the given

Table 4.2: GA parameters

description	symbol	value(s)	comment
Chromosomes in the population	n_{chrom}	240	constant
Chromosomes for elitism	n_{elite}	2	
Initial number of genes	n_{genes}	4	applies to each chromosome
Max n of iterations	n_{iter}	4000	
Stopping fitness threshold	fit_{stop}	0.95	
Crossover probability	p_{cross}	0.7	
Mutation probability	p_{mut}	0.16	applies to values and nodes
Mut. insertion probability	p_{mut}	0.01	
Mut. replication probability	p_{rep}	0.01	
Mut. deletion probability	p_{del}	0.02	
Mut. acceptance probability	p_{acp}	$1 - f_{bc}$	uses fitness of best chromosome

margin), as is shown in Figures 4.6a and 4.6b that represent the desired and the achieved signals (respectively). Comparing these signals, it stands out that there is an offset in V_o (of 511 mV), but the existence of the offset was deliberately allowed in the solution of this problem and taken into account in the fitness function.

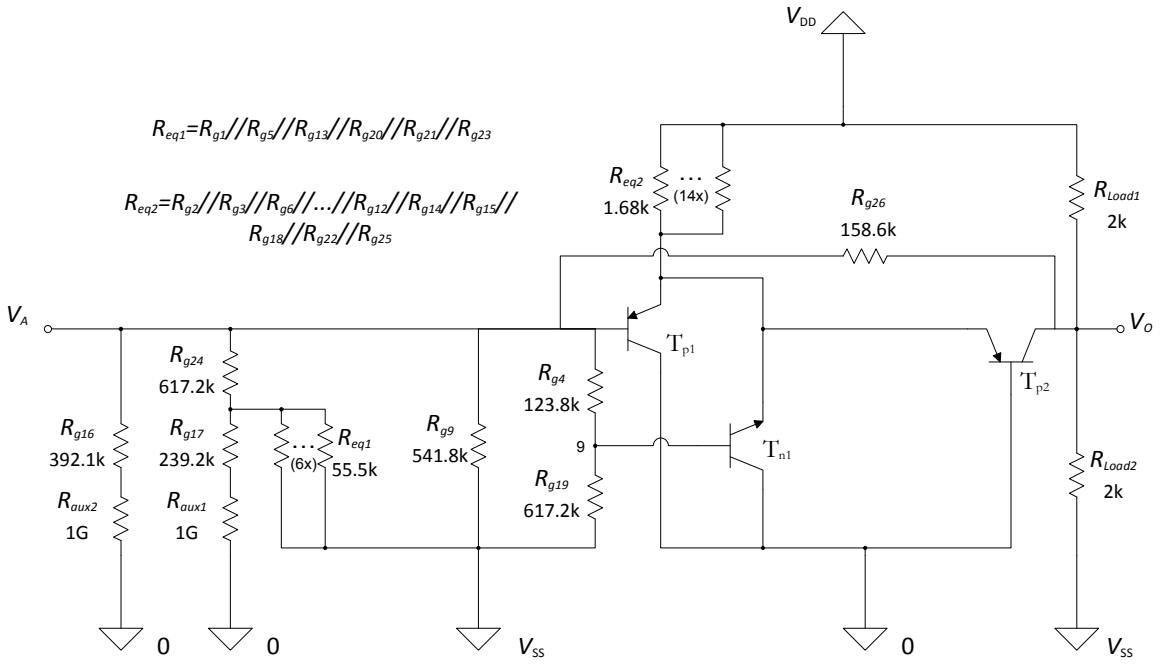
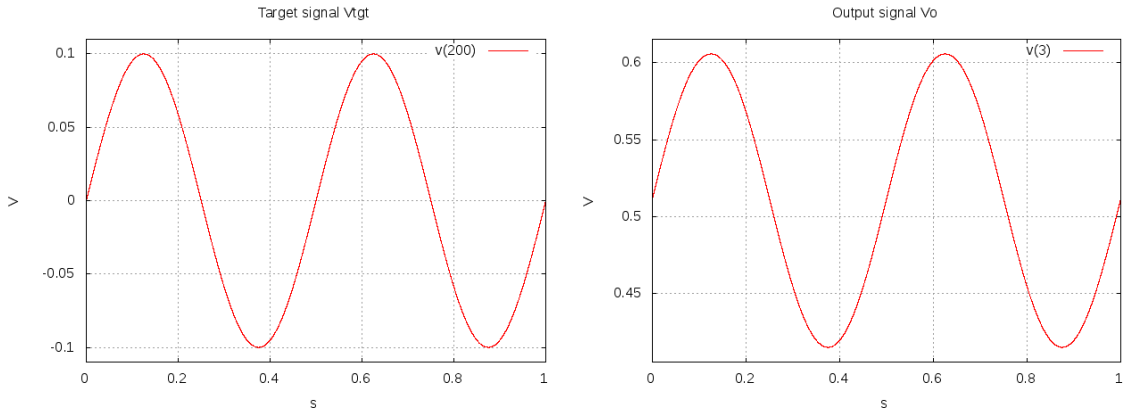


Figure 4.5: Circuit obtained at iteration 1222.



(a) Desired (target) (b) Output

Figure 4.6: Desired (target) and output signals

4.1.1.2 Fitness function

The primary goal of this test was to assess the ability of the GA to generate an amplifier with gain 10, so few other specifications were considered. The intended output swing was rather low, just 200 mV, and there was not any specifications regarding dynamic characteristics, like bandwidth or slew rate. However, four constraints were used: three hard constraints, regarding the current in power supply sources (V_{DD} and V_{SS}) and the current drawn from input source (V_A), and a soft constraint, regarding the number of components in the circuit, whose weight is adapted along the evolution of the GA.

The global fitness function used by the GA is given by eq. 4.1, where the partial functions f_{V_O}, f_{I_A}, $f_{I_{DD}}$, $f_{I_{SS}}$ and $f_{n_{genes}}$ have the form of eq. 4.2. They all are bounded to interval $[0.0\,,1.0]$, and they are all merit functions which means fitness grows towards 1.0 as *achieved* decreases towards *desired*. These functions are described in Table 4.3.

$$f_{glb} = f_{V_O} \times f_{I_A} \times f_{I_{DD}} \times f_{I_{SS}} \times f_{n_{genes}} \tag{4.1}$$

$$f = 1 - e^{-weight^{-1} \times (desired/achieved)} \tag{4.2}$$

In Table 4.3, the *achieved* value $e_{V_{o_{rms}}}$ is a measure of similarity between the actual signal obtained at the output and the desired (or target) signal, and is given by eq. 4.3. In this equation, x_i^a and x_i^d are the i^{th} elements of vectors $\boldsymbol{x}^a$ and $\boldsymbol{x}^d$ produced by NGSPICE for nodes V_o and V_{tgt} of the circuit of Figure 4.3 as a result of the transient analysis requested in the netlist of Figure 4.4 (in this analysis, both vectors have 10000 elements). Value $\widehat{x^a}$ is the average of all elements of vector $\boldsymbol{x}^a$, which is a good estimate of the output offset voltage of the amplifier, which is indifferent (in this test) for the notion of similarity between the actual output signal and the desired one.

Current $I_{A_{rms}}$ (in Table 4.3) is the *rms* value of the current in source V_A, calculated with eq. 4.4.

Table 4.3: Fitness parameters

description	function	weight	desired	achieved
Minimize $e_{V_{o_{rms}}}$	f_{V_O}	0.5	10 mV	$e_{V_{o_{rms}}}$
Minimize $I_{A_{rms}}$	f_{I_A}	0.25	1 µA	$I_{A_{rms}}$
Minimize $I_{DD_{rms}}$	$f_{I_{DD}}$	0.25	2 mA	$I_{DD_{rms}}$
Minimize $I_{SS_{rms}}$	$f_{I_{SS}}$	0.25	2 mA	$I_{SS_{rms}}$
Minimize n_{genes}	$f_{n_{genes}}$	0.005 to 0.01	1 gene	number of genes

$$e_{V_{o_{rms}}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i^a - \widehat{x^a} - x_i^d\right)^2} \tag{4.3}$$

Similarly, $I_{DD_{rms}}$ and $I_{SS_{rms}}$ are the *rms* value of currents in sources V_{DD} and V_{SS}.

$$I_{A_{rms}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i^a\right)^2} \qquad \text{(where vector } \boldsymbol{x}^a \text{ is the current in source } V_A\text{)} \tag{4.4}$$

As mentioned before, the circuit of Figure 4.5 was obtained when the global fitness function f_{glb} reached a stopping threshold of 0.95. Figure 4.7 shows the average fitness of all chromosomes in the population (green line) and the fitness of the best chromosome (red line), up until iteration 1222 of the GA. In this figure it is also shown the low passed version of the average fitness of all chromosomes in the population (gray line), which may be used by the GA to detect stagnation of the evolution. In fact, stagnation is decided by the conjunction of lack of enough increase in both this low passed version of the population average fitness and the best chromosome fitness. As can be seen in this figure, roughly between iteration 600 and iteration 1000 there isn't any visible improvement in either of those fitness values, so stagnation could be wrongly inferred. The number of iterations without fitness improvement that leads to the stagnation decision is another parameter that is not easy to configure: if it is too small, it can lead to early abandonment of some runs of the GA that would eventually yield good results, but if it is too high, it can lead to too much CPU time being wasted on runs that would never yield useful results.

4.1.1.3 Controlling the number of components

In this type of test, the optimum number of components in the final circuit is unknown *a priori*, so it is not known how many components the embryo circuit should (ideally) have; it is a task for the GA to figure out how many components should be used for the fulfillment of the objective. The technique used in this type of test to deal with the number of components in circuits is to set a minimum number of components in the embryo circuit and then ensure that the global parametrization of the GA (namely the probabilities of the

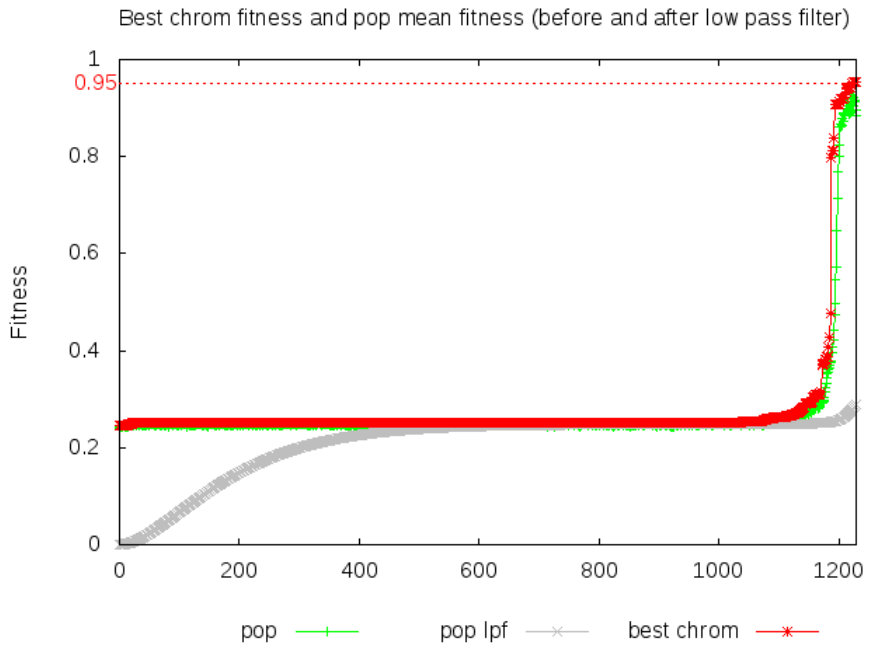


Figure 4.7: Fitness values until iteration 1222.

insertion and deletion features of the mutation operator) creates a drive for component growth. So, it is expected that the GA starts its search using small circuits and gradually progress to circuits with a larger number of components.

However, as the number of components in the circuits increases so does the duration of their simulation. Not only because of the burden of simulating unnecessarily large circuits, but also because it is intrinsic to the purpose of synthesizing a circuit to be able to minimize the number of components used, there is the need for some kind of control over the number of components used in each circuit.

Function $f_{n_{genes}}$ is used to provide some degree of control over the number of genes (components) in each chromosome (circuit). This function penalizes the fitness of the chromosome for the use of components and is difficult to parametrize, mainly because the optimum number of components in the final circuit is unknown. If it penalizes too much, the search space explored by the GA is not sufficiently extended during the evolution, but if it penalizes too less, the number of components grows up to problematical values (a well-known problem often referred to as "bloating" [Žiga Rojec et al., 2019, Zebulum et al., 2000, Mattiussi and Floreano, 2007, Ando and Iba, 2000]). In this test, the *desired* value used was one component, which may be an idealistic value (certainly it is not a realistic one) but works as a minorant, *i.e.*, a "catch all" value. Anyway, the outcome of choosing this value can be tempered by the value chosen for the weight used in function $f_{n_{genes}}$, so that the effect in f_{glb} is the desired one.

In Figure 4.8 it can be seen the evolution of the mean number of genes (components) in the population and the number of genes of the best chromosome for the initial 100 iterations

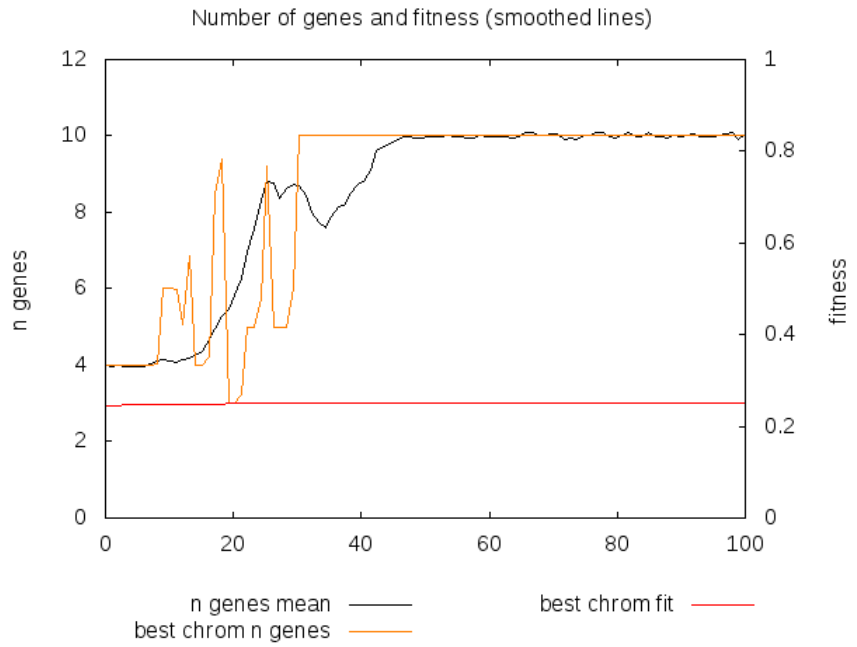


Figure 4.8: Number of genes and best chromosome fitness for the first 100 iterations.

of this test. It is also shown the fitness of the best chromosome, which is quite low (< 0.3) and virtually constant in this initial phase of the GA. As can be observed, there is initially an adjustment that is followed by a steady stage: initially, in about 50 iterations, both the mean number of genes in the population and the number of genes of the best chromosome grow from 4 to 10, and stabilize for the remainder 50 iterations shown in the graphic. As expected, this number of 10 genes reached after iteration 50 is not directly parameterized anywhere in the algorithm; it is an indirect consequence of the weights used in the partial fitness functions and of the remaining parameterization of the GA.

In several tests similar to this one, it was often observed that the largest weight used in function $f_{n_{genes}}$, which still allows a sufficiently extended search, ends up not being penalizing enough to avoid the existence of many chromosomes in the population that still have an excessive number of components (the majority of this components won't be useful in the final circuit). So, in order to try to get rid of part of these components in a later stage of the circuit's evolution, this weight is adapted during the evolution so that it starts with a low value (to allow an extended search) and ends with a higher value (which achieves a "cleaning" effect of the circuit by removing unnecessary components). In this test, this adaptive mechanism was steered by the fitness of the best chromosome in the population, using a non linear function like the one depicted in Figure 4.9.

Although this technique removes many unnecessary components in the late phase of evolution, the final circuit is usually still left with several useless or redundant components, as can be seen in Figure 4.5. It is often useful to extend the number of iterations after the stopping fitness threshold is reached in order to achieve an extra "cleaning" effect upon

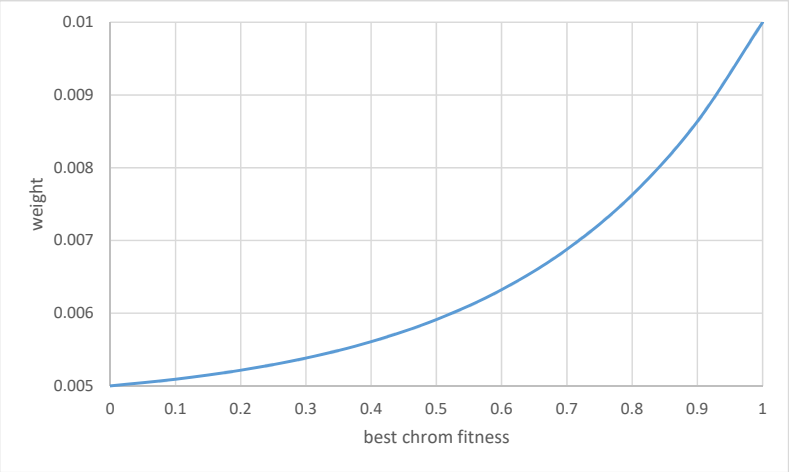


Figure 4.9: Mapping best chromosome fitness to fn_{genes} weight.

the final circuit that is still able to remove some unnecessary components. This extension to the number of iterations was performed over the circuit of Figure 4.5 and the circuit obtained (at iter 1535) is shown in Figure 4.10.

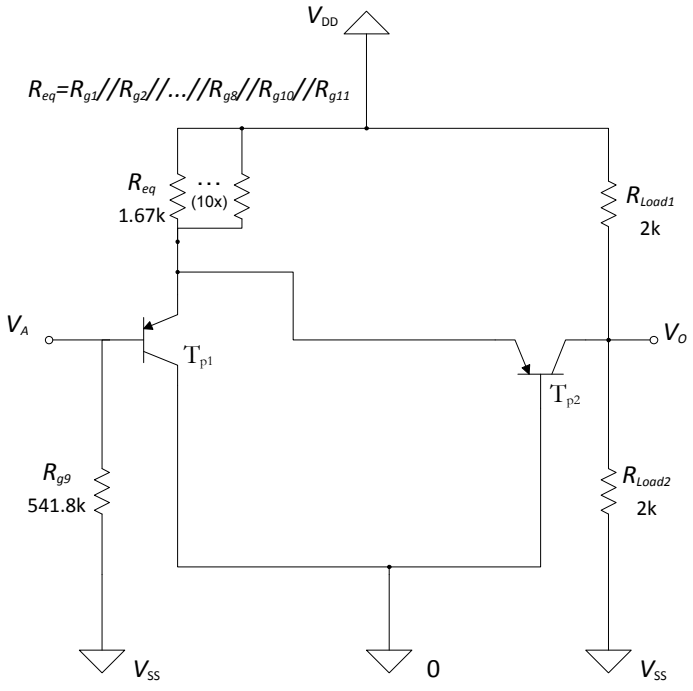


Figure 4.10: Circuit obtained at iteration 1535.

In this circuit there are still some redundant components but most that were present in Figure 4.5 have been removed. Fitness evolution until iteration 1535 is shown in Figure 4.11, where it can be seen that both the fitness of the best chromosome and the average fitness of the population increased slightly after iteration 1222, which was mainly due to the component removal effect. The GA was stopped when the best chromosome fitness stopped improving for more than 100 iterations.

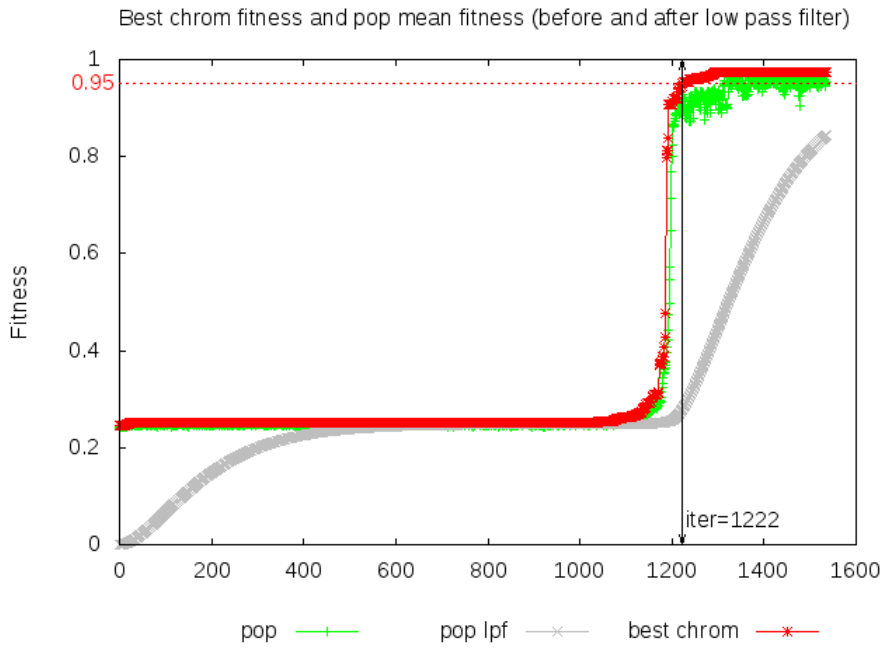


Figure 4.11: Fitness values until iteration 1535.

The number of genes (both for the best chromosome and for the average in the population) and the fitness of the best chromosome are shown in Figure 4.12 for the entire evolution of the circuit. After about 50 initial iterations (initial transient of the GA's evolution), both the number of genes of the best chromosome and the average number of genes in the population remained in a steady value of (approximately) 10 until iteration 975.

At iteration 976 there was a very small increase in the fitness of the best chromosome (unnoticeable in the graphic but easily spotted in raw data) and the number of genes of the best chromosome suddenly increases to 14 and keeps increasing, closely followed by the average number of genes in the population, and later followed by a significant increase in the fitness of the best chromosome, until a maximum of 47 genes is reached. By the time this maximum is reached, the fitness of the best chromosome is still at a modest value of 0.48 but it is enough to moderate the growth momentum of the number of genes, which soon begins to decrease while the best chromosome fitness remains increasing, until it reaches 0.95, which was used as a stopping threshold in early tests. But in this test the GA was allowed to continue for a little longer, and the result is that the number of genes of the best chromosome stabilized in 13 and its fitness even increased slightly, up to 0.97.

In some of the previous figures, graphical information was given about the average number of genes (per chromosome) in the population, but no information was given about the variation of the maximum and minimum values of the number of genes (per chromosome) in the population throughout evolution (to avoid overlapping too many lines in the graphs). Figure 4.13 now shows how the maximum and minimum are related to the mean during the evolution of the GA. It also shows the standard deviation of the number of genes, which

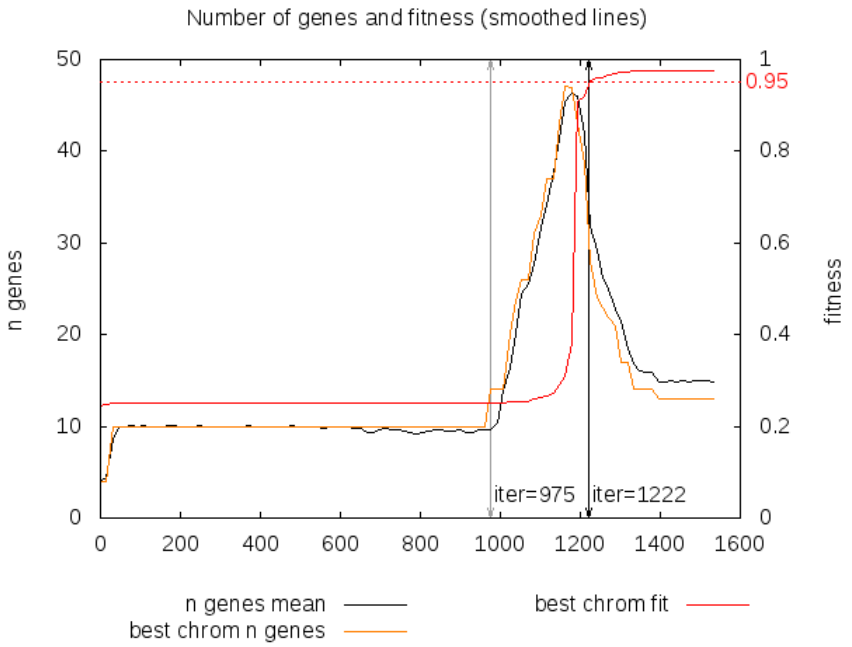


Figure 4.12: Number of genes and best chromosome fitness.

is quite low for most of the time, suggesting that there is significant concentration around the mean and that there are few chromosomes with maximum or minimum number of genes.

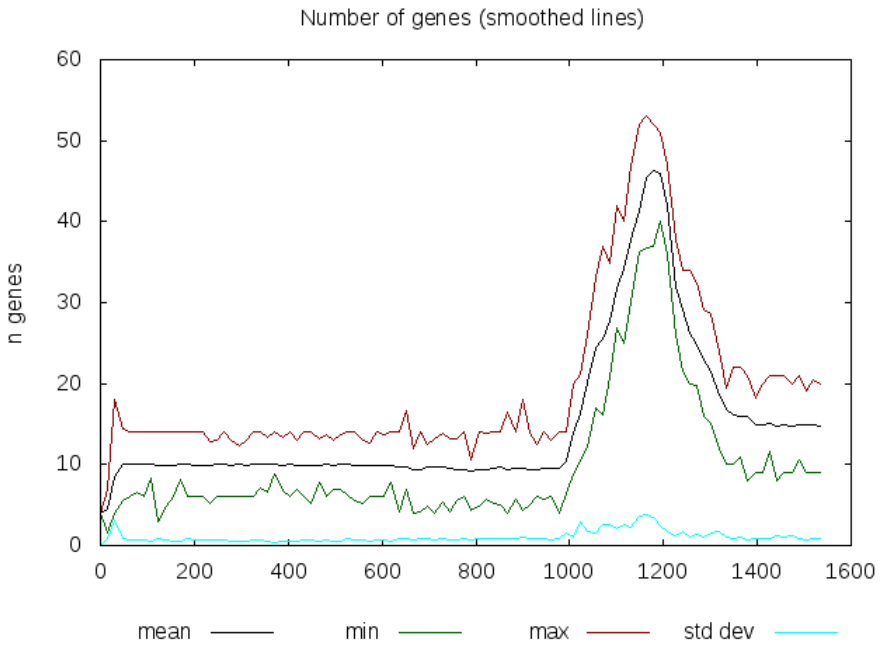


Figure 4.13: Fitness values until iteration 1222.

4.2 Revisiting digital circuits

While the previous test was in progress, other tests were being carried out, namely in some of the logic circuits already tested (some of which have already been successfully synthesized, but others were never successfully synthesized by GA). Using the adaptive parameters approach (continuous adaptive mode) complemented with multiphase synthesis (discrete adaptive mode), some logic circuits were synthesized, like a 2-input NAND gate and a 2-input XOR gate, and these results are now briefly presented.

Unlike the previous test described in Section 4.1.1 (where no phases were used), the tests described here used three phases (as described in Section 3.4.3), which can be labeled "Topology generation phase", "Component minimization phase" and "MOS area minimization phase". Although the first phase is being labeled as "Topology generation", in fact it comprises topology generation and component sizing, and in this phase the GA is expected to generate a circuit that satisfies the logic gate truth table without much concern for optimizing other characteristics of the circuit that are considered of secondary importance in this phase, such as the number of components and the total area used in MOS transistors. As in the previous test, this "low concern" characteristic is achieved by limiting the weights applied to the constraints that evaluate the number of components and the total area used in MOS transistors. These weights are still adapted as a function of the best chromosome fitness, but are bounded with heuristic values in such a way that the upper bound is not so high as to prevent the development of a viable circuit that complies with the desired characteristic function (the gate truth table), and the lower bound is not so low as to interfere too much with the evolution of the algorithm, namely by letting the number of components grow too much and compromising the ability of the GA to converge to a useful solution within a reasonable time frame.

For the synthesis of the NAND gate, phase 0 lasted until iteration 101 and the circuit shown in Figure 4.14 was generated. This circuit conforms to the truth table for a 2-input NAND gate, although it certainly uses much more transistors than is actually needed to accomplish this objective. The criterion used to stop this phase was not stagnation, but the achievement of a fitness of 0.95 by the best chromosome, so it is not unexpected that the circuit still has redundant or useless components. With this stopping criterion, a very deep "cleaning" effect on the number of components is not expected, being transferred to the next phase.

Phase 1, which was dedicated to the minimization of the quantity of components, lasted until iteration 958, and produced the circuit shown in Figure 4.15a, which yields the classic topology of this gate. As expected, and unlike the previous phase, this phase had a deep "cleaning" effect on the number of components, and removed all the redundant or useless transistors and resistors.

The last phase, dedicated to the minimization of the total MOS area used by the transistors

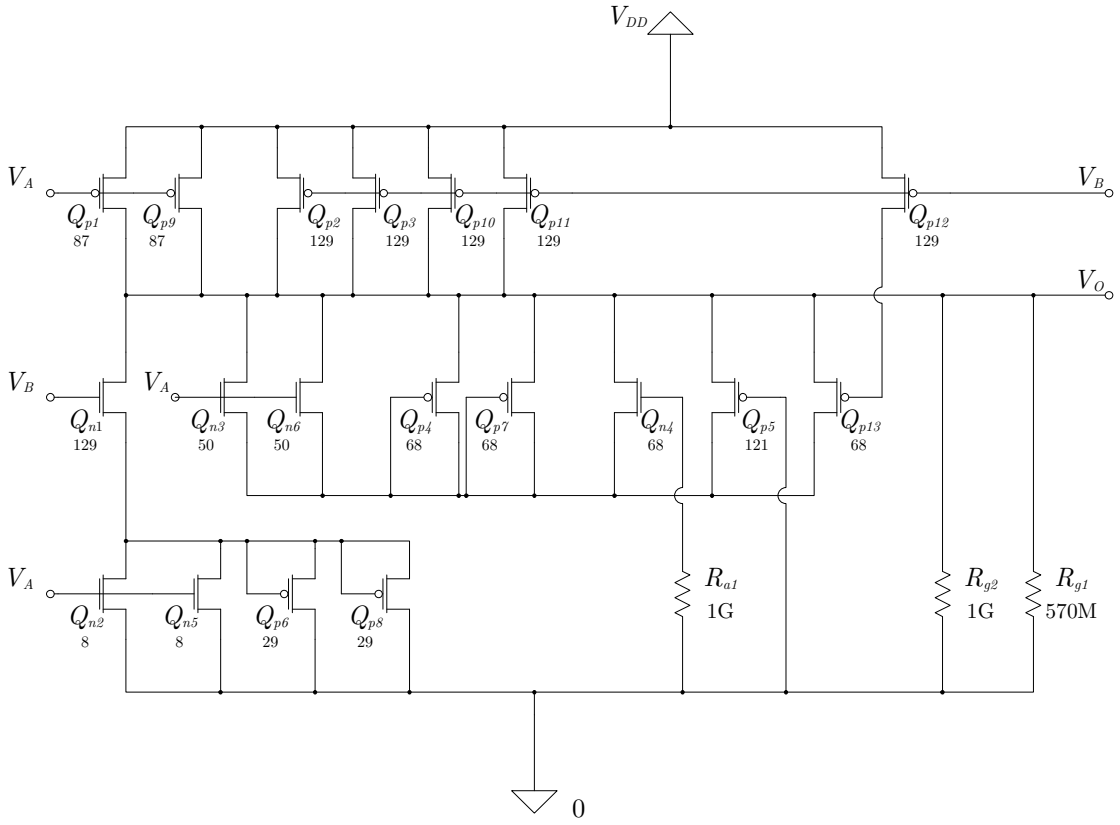
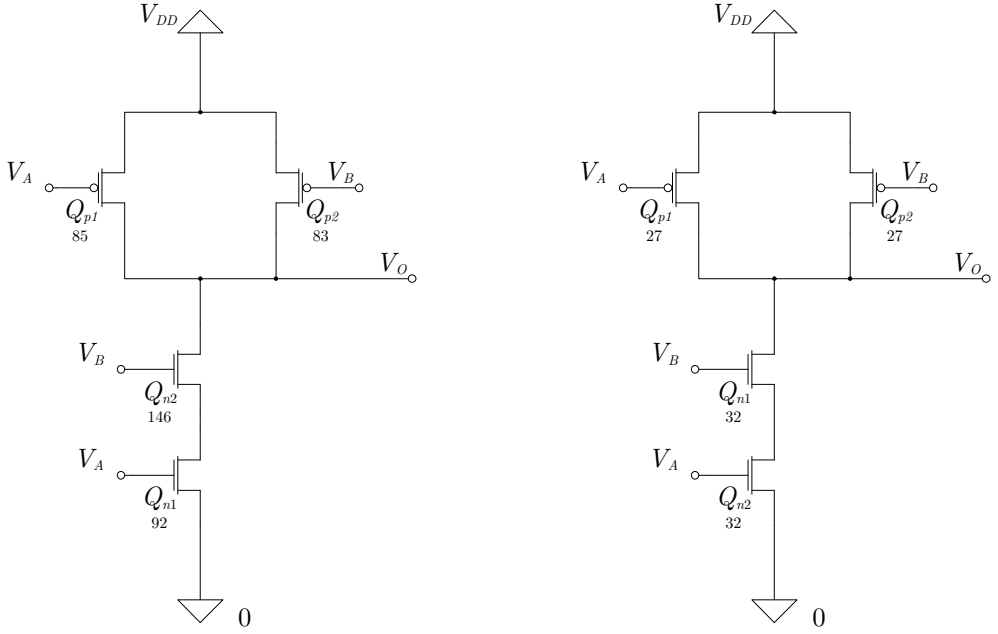


Figure 4.14: Nand generated at the end of phase 0.



(a) End of phase 1

(b) End of phase 2

Figure 4.15: NAND gate generated at the end of phases 1 and 2.

of the circuit, lasted until iteration 1503 and produced the circuit shown in Figure 4.15b. In this figure it can be observed that the topology of the circuit has not changed but the size of all transistors was reduced.

It is true that circuits similar to the latter had already been obtained (see Figure 3.16 on page 54) without resorting to the techniques that have now been used in the synthesis of this circuit. But previously, a NAND port made with 4 transistors was only obtained when the total number of transistors available to the GA was exactly 4, whereas now, although the number of transistors available is virtually infinite, the GA ends up producing a circuit with the same 4 transistors.

Another circuit that has never been successfully synthesized by the GA until now is the 2-input XOR gate. Although many hours of CPU time have been invested in attempts to synthesize this circuit, it has never been possible to obtain such a circuit while using FLCs. Of course, this could just be an indicator that not enough hours were spent in this endeavor, or that the right number of transistors in the embryo circuit was simply not reached, but these tests were stopped when it was considered that the limit for a reasonable investment (in research and CPU time) had already been exceeded.

However, applying the same techniques used in the previous test (2-input NAND gate), the GA generated the circuit shown in Figure 4.16, at the end of phase 3, after 9316 iterations. This circuit complies with the truth table of a 2-input XOR gate, and although it is much more difficult to obtain than a NAND gate, as indicated by the fact that phase 0 ended after 6147 iterations (as opposed to only 101 iterations for the NAND synthesis), it expresses the difference in the synthesis capacity that the GA has when comparing the use of VLCs versus the use of FLCs.

4.3 Amplifiers using MOS transistors with $L = 10\,\mu m$

In the set of tests described in this section, a *LEVEL 1 MOSFET* SPICE model was used for the simulation of all MOS transistors, mainly because of the time penalty for using higher level models. This choice entails a trade-off between accuracy and simulation time. As referred in [SYNOPSYS, 2010, pp. 8–9], *LEVEL 1 MOSFET* models offer low simulation time and a relatively high level of accuracy for timing calculations but for higher precision results more detailed models should be used. Or, as clearly stated in [SYNOPSYS, 2010, pp. 68–69]: *Use the LEVEL 1 MOSFET model if accuracy is less important to you than simulation turn-around time. (...), LEVEL 1 run-time can be about half that of a simulation using the LEVEL 2 model.*

In turn, the use of *LEVEL 1 MOSFET* model implies the use of transistors with a channel length of no less than 10µm, in order to avoid an increased loss of precision [Allen and Holberg, 2012, pp. 68] [Patel, 2014].

Although current MOSFET technologies provided by well-known foundries support tran-

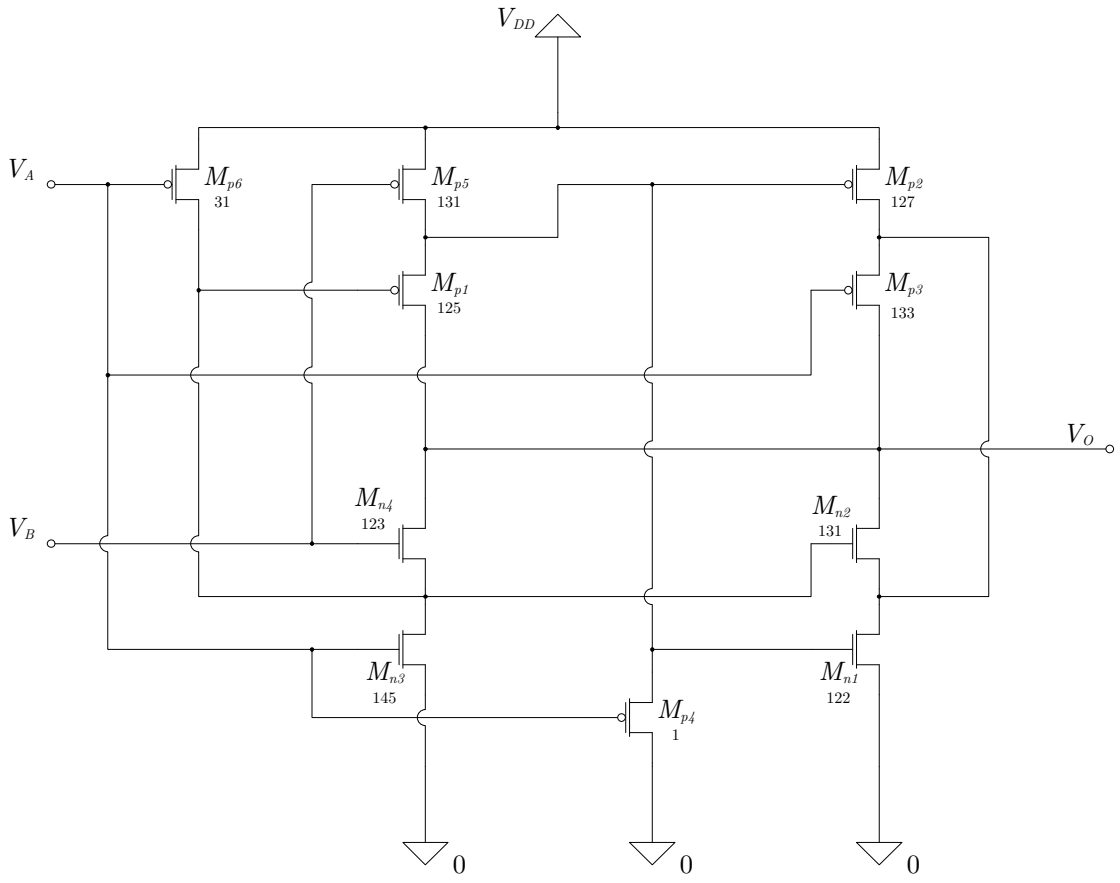


Figure 4.16: XOR gate generated by the GA.

sistor lengths much smaller than 10 µm, the usage of *LEVEL 1 MOSFET* model dictated that $L = 10\,\mu m$ should be used in all tests in this section.

4.3.1 A 20dB DC Amplifier

Like the goal of the test described in Section 4.1.1, the goal of this test was the synthesis of a DC amplifier with a gain of 10, but instead of using bipolar transistors, MOSFET transistors were used. Although the GA is essentially the same as the one used in the previous tests, some additional techniques were introduced, like those described in Section 3.4.

4.3.1.1 The embryo circuit

The embryo circuit shown in Figure 4.17 was used for this test. This circuit is similar to the embryo circuit used for the amplifier described in Section 4.1.1 (Figure 4.3), but this one uses MOS transistors and provides a bias network, available to the evolvable circuit, consisting of transistors M_{pbias} and M_{nbias} and a current source, I_{bias}. The evolvable circuit starts with two random resistors (random in value and in connectivity) and two sets of five MOSFETs each, all with random W/L ratios and random connectivity. Some parameters of this circuit can be found in Table 4.4.

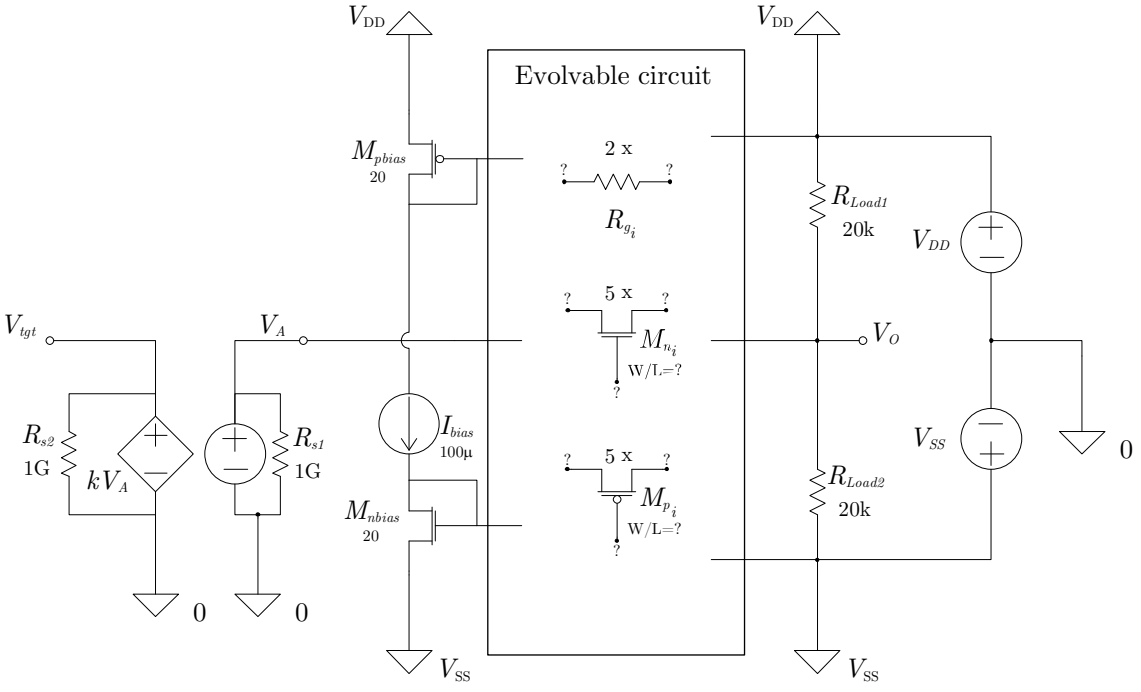


Figure 4.17: Embryo circuit for a DC amplifier with gain 10 using MOS transistors.

The parameters used in this test's embryo circuit are similar to those used in the previous test (shown in Table 4.1), but differ in the amplitude of the input signal, in the maximum number of nodes and in the fact that the transistors, in addition to being of the MOSFET type instead of being bipolar, now have a variable attribute, which is the W of each transistor (the L is fixed and the same for all transistors). In this test, the amplitude of the input signal was increased so that a higher output swing of $2\,V_{PP}$ was achieved. The

maximum number of nodes that the GA can use to evolve the circuit was increased from 10 to 14 because it was presumed that this circuit could need more nodes, if not in its final version, eventually in its intermediate versions. The W attribute of transistors is implemented by an extra real variable (an allele) of the MOS transistor descriptor (gene), and the GA needs to search and optimize this attribute for each transistor.

Table 4.4: Circuit parameters

Positive power supply	V_{DD}	2.5 V
Negative power supply	V_{SS}	−2.5 V
Target signal source gain	k	10
Input signal $v_A(t)$	$A\sin(\omega t)$	$A = 100\,\text{mV}$, $\omega = 2\pi \times 2\,\text{Hz}$
Transistors W/L ratio	W/L	1 to 200
Resistors in evolvable circ	$R_{g1}, R_{g2}..R_{gN}$	10 Ω to 1 MΩ
Max n of nodes in evolvable circ	$node_{max}$	14 (excluding ground)

4.3.1.2 Fitness function

The global fitness function used in this test is similar to the one used in the previous test (eq. 4.1) but with an extra parcial function, as shown in eq. 4.5. The goal in using this new function $f_{mos_{area}}$ is to optimize the total area used by transistors, which is approximated in this calculus by summing the channel area, given by the WL product, of each transistor.

$$f_{glb} = f_{V_O} \times f_{I_A} \times f_{I_{DD}} \times f_{I_{SS}} \times f_{n_{genes}} \times f_{mos_{area}} \tag{4.5}$$

4.3.1.3 Additional techniques

Although this test has several similarities to the previous one, it is distinctive in some characteristics, one of which is the usage of some of the techniques described in Section 3.4.

One such technique is a detector of parallel components that may be used to accelerate the simplification of circuits. Although the parallel association of components can still be done intrinsically by the GA, there is now a procedure to specifically detect and associate parallel components that may be used during the execution of the GA.

In this test, the circuits evolved by the GA were simulated using the test bed technique described in Section 3.4.1. As a source of pseudo-noise, a simple series grouping of two sinusoidal signals was used. This simplification results from a compromise between simulation time and the intended effect of avoiding floating transistor gates.

Another technique used in this test is the use of phases (or stages) with distinct parametrizations during the evolution of the circuit. These distinct parametrizations allow for some steering of the GA in order to direct its search and optimization efforts towards a specific

optimization target, such as the total number of components or the total area used in MOS transistors.

Along with the parallel association of components and the use of phases, this test also used a variable number of chromosomes in the population. This was done primarily to balance the execution time with the search capabilities of the GA. Although the initial size of the population is 240 chromosomes, as in the previous test, this size may change during evolution in this test.

Multiphase parametrization

One technique used in this test is multiphase parametrization, which was applied in a sequence of three phases, as described in Section 3.4.3. Fitness evolution for all phases is shown in Figure 4.18, where phase transitions are marked by vertical arrows (at iteration 2017 and iteration 10364). Some fitness parameters common to all phases can be found in Table 4.5, while others used for phase 0 can be found in Table 4.6.

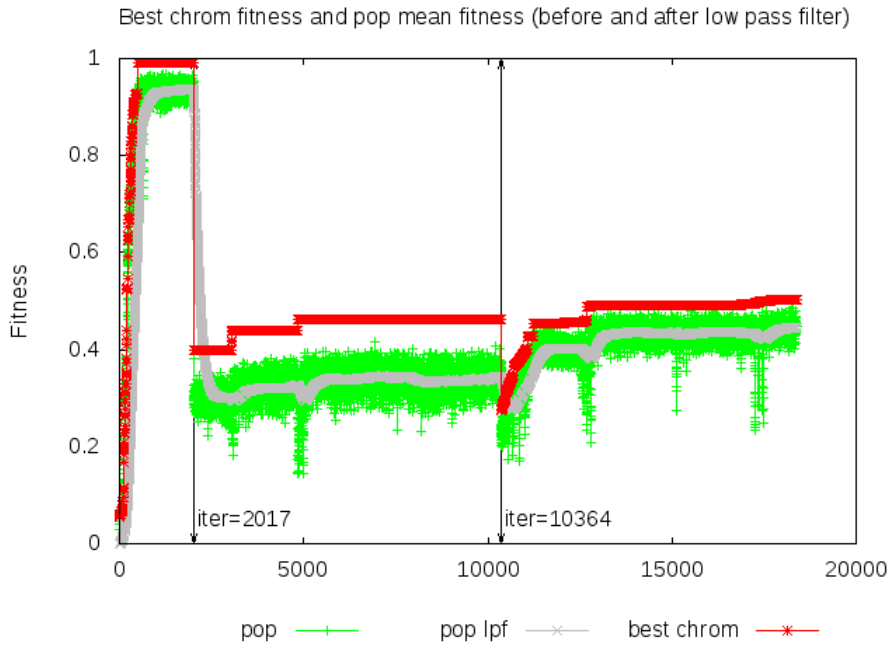


Figure 4.18: Fitness values until iteration 18380.

Table 4.5: Fitness parameters common to all phases

Description	Function	Weight	Desired	Achieved
Minimize $e_{V_{O_{rms}}}$	f_{V_O}	0.5	20 mV	$e_{V_{O_{rms}}}$
Minimize $I_{A_{rms}}$	f_{I_A}	0.25	0.1 µA	$I_{A_{rms}}$
Minimize $I_{DD_{rms}}$	$f_{I_{DD}}$	0.25	4 mA	$I_{DD_{rms}}$
Minimize $I_{SS_{rms}}$	$f_{I_{SS}}$	0.25	4 mA	$I_{SS_{rms}}$

Although all weights shown in Table 4.5 are equal to those used in the previous test

(Table 4.3), the desired values have been changed to match the characteristics of this test. Because of the higher output swing intended in this test, the desired values for currents $I_{DD_{rms}}$ and $I_{SS_{rms}}$ were increased, as was the error $e_{V_{o_{rms}}}$ (although it was not increased proportionally).

All four weights shown in Table 4.5 are fixed during the evolution of the GA. Instead, weights shown in Table 4.6 are changed along the evolution of the GA, according to mapping functions like the one depicted in Figure 4.9. These weights start with a relatively low value which is increased when there is enough growth of the best chromosome fitness. However, it is not easy to combine both intended effects simultaneously in the GA: minimizing the number of components and the total MOS area used. Therefore, an attempt was made to dissociate the application of the two effects, applying them sequentially: this was done in phases 1 and 2.

Table 4.6: Fitness parameters for phase 0

description	function	weight	desired	achieved
Minimize n_{genes}	$f_{n_{genes}}$	0.001 to 0.004	1 gene	number of genes
Minimize mos_{area}	$f_{mos_{area}}$	0.015 to 0.010	*technology dependent*	$\sum(W_i L_i)$

Using a stop criterion of stagnation of the GA, the first phase terminated in iteration 2017 (Figure 4.18). Then, in phase 1, an increment of 0.1 was done to the last value of the weight of function $f_{n_{genes}}$, while the weight of function $f_{mos_{area}}$ was kept equal to the last value used in previous phase, as shown in Table 4.7. And then the GA was allowed to evolve until stagnation was reached again.

As seen in Figure 4.18, that change in the weight of function $f_{n_{genes}}$ leads to a sudden decrease in fitness that should be recovered later, which indeed occurred in this test after the significant period of 8347 iterations.

Table 4.7: Fitness parameters for phase 1

description	function	weight	desired	achieved
Minimize n_{genes}	$f_{n_{genes}}$	0.104	1 gene	number of genes
Minimize mos_{area}	$f_{mos_{area}}$	0.010	*technology dependent*	$\sum(W_i L_i)$

After reaching stagnation, phase 1 was terminated and the weight of function $f_{mos_{area}}$ was increased by 0.1 while the weight of function $f_{n_{genes}}$ was kept unchanged, as shown in Table 4.8. And again, during phase 2, the GA was allowed to evolve until stagnation was reached, which occurred after 8467 iterations (in iteration 18831), dictating the end of this phase (and the end of this run of the GA). Similarly, in the transition from phase 1 to phase 2, the change in the weight of function $f_{mos_{area}}$ caused a sudden (but not so large) decrease in fitness, which was recovered later (see Figure 4.18, after iteration 10364).

Table 4.8: Fitness parameters for phase 2

description	function	weight	desired	achieved
Minimize n_{genes}	$f_{n_{genes}}$	0.104	1 gene	number of genes
Minimize mos_{area}	$f_{mos_{area}}$	0.110	*technology dependent*	$\sum(W_i L_i)$

Other adjustments

Changing weights are not the only parameter adjustments made on a per-phase basis. Other adjustments can be made, and in this test some of the GA parameters were adjusted in each phase transition, such as the stopping criterion (which does not have to be the same for all phases), the population size (which can be variable, and has a "preferred" value for each phase), the probability of the mutation-deletion operator (which can be adjusted according to the purpose of the phase), and the periodicity used to execute the parallel association detector (whose relevance is not the same for all phases).

A list of parameters common to all phases can be found in Table 4.9. In this table it can be seen that in this test there was no limit to the number of iterations and that the stopping criterion for each phase was stagnation. In fact, having no limit on the number of iterations means that stopping the execution of a particular run of the GA is done manually when the execution time exceeds what is considered reasonable for that particular research effort (this is a very subjective issue, depending on human factors such as the willingness to wait, which is closely related to the urgency of obtaining results, among other factors, and also on objective factors such as the availability of computer power).

Using only stagnation as a stopping criterion means that the transition to the next phase is made without any requirement of reaching a minimum fitness threshold. This seems to be a good criterion for ending phases 1 and 2. But for phase 0 it is only useful if it is accompanied by close human monitoring of fitness evolution, leading to early termination of execution if there is a transition from phase 0 to phase 1 without sufficient fitness achievement. A better (and autonomous) criterion for stopping phase 0 seems to be the conjunction of reaching a minimum fitness threshold and fitness stagnation.

The parameters shown in Table 4.9 are not phase-dependent, so remain unchanged for all phases, but other parameters are phase-dependent, like those found in Table 4.10. As can be seen in this table, the initial size of the population is 240 chromosomes and the preferred size of the population is also 240, which means that the GA starts with this population size and will try to regain this same size whenever the number of chromosomes in the population differs.

In this test, the number of chromosomes may increase each time the parallel association algorithm is executed, which can (at most) double the population size (if all chromosomes have at least one pair of components that can be associated). Whenever the population size is above its preferred value, the GA starts to "lose" some chromosomes (usually two in each

Table 4.9: GA parameters common to all phases

description	symbol	value(s)	comment
Chromosomes for elitism	n_{elite}	2	
Initial number of genes	n_{genes}	10	applies to each chromosome
Max n of iterations	n_{iter}	infinite	
Stopping fitness threshold	fit_{stop}	*not used*	stagnation used instead
Crossover probability	p_{cross}	0.7	
Mutation probability	p_{mut}	0.16	applies to values and nodes
Mut. insertion probability	p_{mut}	0.01	
Mut. replication probability	p_{rep}	0.01	
Mut. acceptance probability	p_{acp}	$1 - f_{bc}$	uses fitness of best chromosome

iteration), "losing" those that have the worst fitness in the population. This decrease in population size continues until the preferred value is reached again.

Table 4.10: GA parameters for phase 0

Description	Symbol	Value(s)	Comment
Chromosomes in population	$(n_{chrom})_{ph0}$	240 to 480	init val. 240; preferred val. 240
Mut. deletion probability	$(p_{del})_{ph0}$	0.02	
Parallel assoc. periodicity	$\left(T_{pad}\right)_{ph0}$	500 iter	

Table 4.10 also shows the probability of the deletion feature of the mutation operator used in phase 0, $(p_{del})_{ph0}$, and the periodicity used to execute the parallel association detector, $\left(T_{pad}\right)_{ph0}$. Too much use of the parallel association detector can be quite expensive in execution time of the GA because of the effect on population size and thus in circuit simulation time, so this periodicity results from a balance between the intended effect of the detector and the overall execution time of the GA.

In the next phase, all the parameters shown in Table 4.10 are modified and their new values are shown in Table 4.11. The preferred population size is reduced by half, which

Table 4.11: GA parameters for phase 1

Description	Symbol	Value(s)	Comment
Chromosomes in population	$(n_{chrom})_{ph1}$	120 to 480	preferred val. 120
Mut. deletion probability	$(p_{del})_{ph1}$	0.12	$(p_{del})_{ph1} = 6 \times (p_{del})_{ph0}$
Parallel assoc. periodicity	$\left(T_{pad}\right)_{ph1}$	100 iter	

significantly reduces execution time without sacrificing much of GA's exploration capabil-

ities, considering the intended objective of this phase, which is to reduce the number of components.

The probability of the mutation-deletion operator used in this phase is increased six times, in order to speed up the process of eliminating useless components. Also, with the same goal, the periodicity of the parallel association detector is significantly reduced to 100 iterations.

When this phase stagnates and is terminated, phase 2 starts, and an adjustment is made to the probability of the mutation-deletion operator, reducing it to half of its value used in phase 0, as shown in Table 4.12. The purpose of this phase is to reduce the total area used in MOS transistors, and this is done after useless components have been removed from the circuit, so it is expected that there will be no need to remove more of them. Therefore, it is expected that reducing this probability will not interfere with the main objective of this phase, while at the same time it is expected to increase the execution speed of the GA.

Table 4.12: GA parameters for phase 2

description	symbol	value(s)	comment
Chromosomes in population	$(n_{chrom})_{ph2}$	120 to 480	preferred val. 120
Mut. deletion probability	$(p_{del})_{ph2}$	0.01	$(p_{del})_{ph2} = 0.5 \times (p_{del})_{ph0}$
Parallel assoc. periodicity	$\left(T_{pad}\right)_{ph2}$	100 iter	

The preferred population size in this phase is kept small, at 120 chromosomes, because it is expected that there will be no topology changes in the circuit, only component sizing, and this can be accomplished in small populations. The parallel association detector has not been turned off, but it is expected to have little or no effect in this phase, since its purpose is expected to have been fully served in the previous phase.

4.3.1.4 Synthesized circuits at the end of each phase

Using the parametrization described above, the circuit implemented by the best chromosome was chosen as the outcome of the GA after execution of phase 2. At the end of phases 0 and 1 there is also one chromosome that has the best circuit generated in that phase. Analyzing the best circuit of each phase clarifies the contribution of each phase to the final result.

Circuit generated in phase 0

The GA generated the circuit shown in Figure 4.19 at the end of phase 0 (in this circuit the W/L ratio of each transistor is shown next to the transistor symbol, near the transistor naming label). This circuit already fulfills the main objectives of this test, namely the similarity between the output signal obtained and the desired one, and the current limitations imposed on the power and input voltage sources. However, it is not optimized either in terms of the number of components or the area occupied by the transistors.

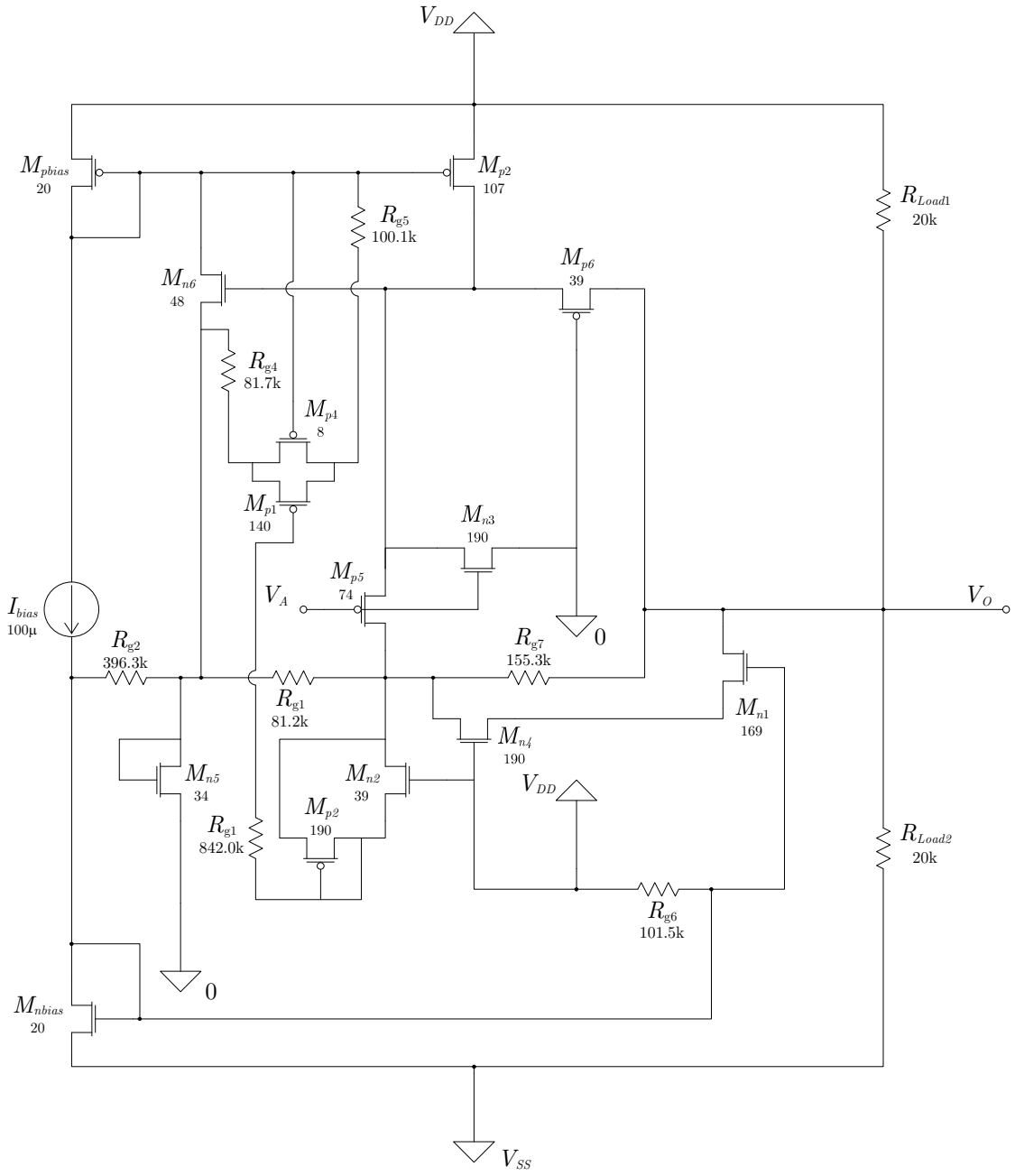


Figure 4.19: DC amplifier with gain 10: circuit generated at the end of phase 0.

Figure 4.20 shows the evolution of the number of genes of the best chromosome along with its fitness and the average number of genes used in the population. In this graph, there are no visible changes in both the number of genes of the best chromosome and its fitness after iteration 504. However, in the previous hundred or so iterations, the fitness of the best chromosome is already high enough to cause a significant increase in the weight of function $f_{n_{genes}}$ (set by its mapping function). Having a relatively high value (considering this phase context) for this weight during most of the evolution period shown in Figure 4.20 implies that this phase already benefits from some of the "cleaning" effect provided by function $f_{n_{genes}}$.

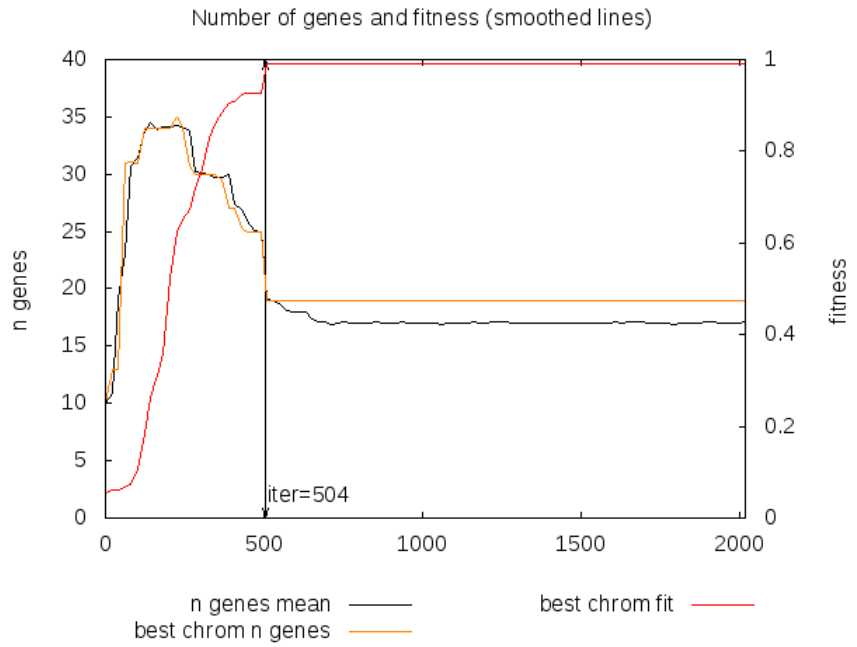


Figure 4.20: Number of genes and best chromosome fitness in phase 0.

After iteration 504 there was no change in the number of genes of the best chromosome, which suggests that, knowing that there are still useless components in the circuit, that the upper limit of function's $f_{n_{genes}}$ weight could be higher than the one used in this phase. Nevertheless, the next phase is intended to address this issue of removing useless components remaining in the circuit, so this upper limit was left unchanged for other similar tests.

Circuit generated in phase 1

Although there aren't parallel components in the circuit of Figure 4.19, there are still useless components in this circuit. The next phase, phase 1, managed to eliminate some of them (two transistors and three resistors), producing the circuit shown in Figure 4.21.

This circuit has less components and still keeps its level of accomplishment regarding not only the similitude between its output and the target signal but also regarding current consumption from power and input sources. To describe how this was achieved, in the context of this explanation the reduction of components will be referred to as the *secondary*

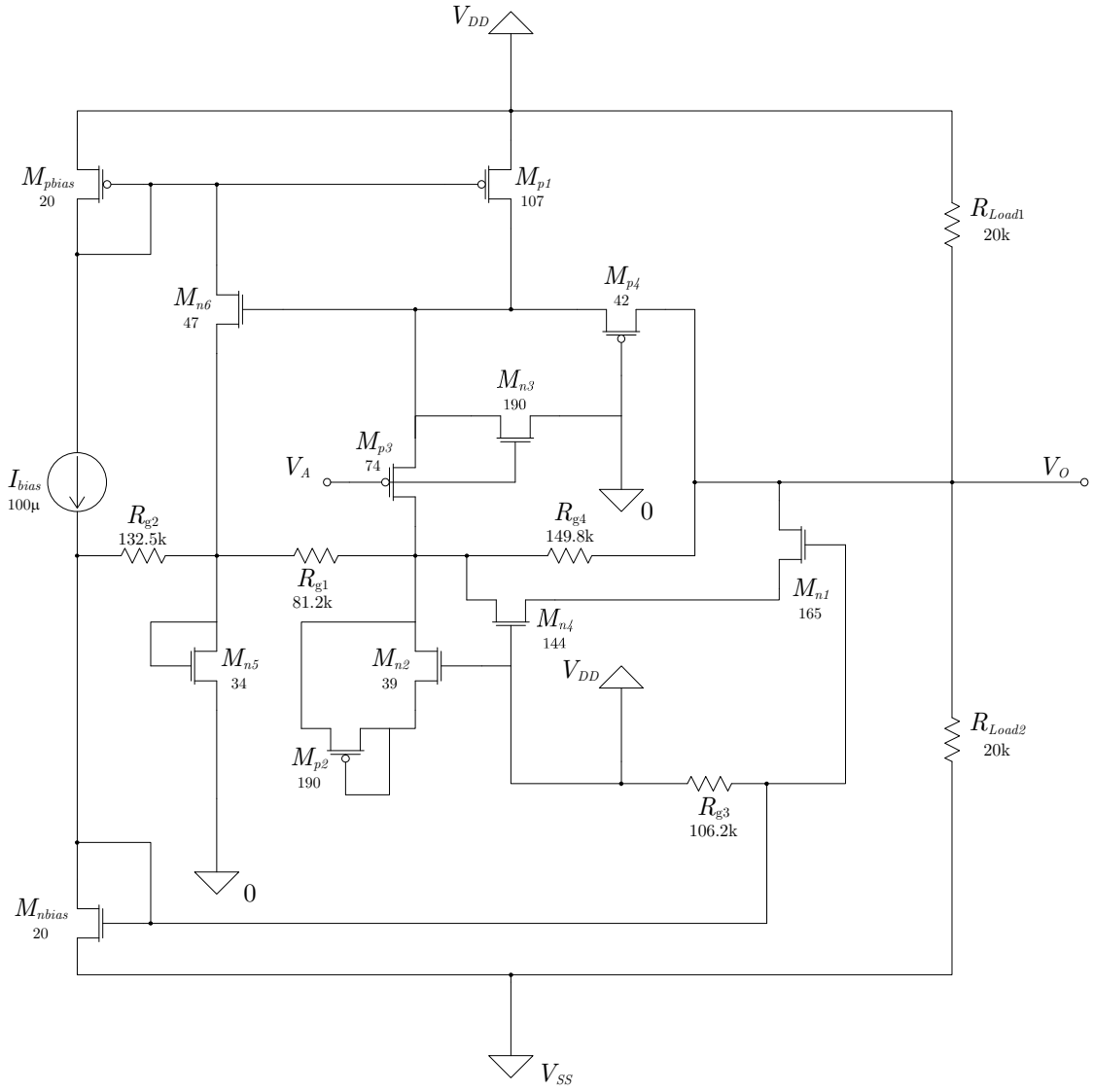


Figure 4.21: DC amplifier with gain 10: circuit generated at the end of phase 1.

objective of this phase, while the similitude between the circuit's output and target signal, together with its current consumption from power and input sources, will be referred to as the *main objective* of this problem. The feature of maintaining the fitness related to the *main objective* while being able to reduce the number of components, *i.e.*, while being able to increase the fitness related to *secondary objective* being addressed in this phase, was not managed directly by the choice of weights of all partial fitness functions. Instead, it was managed by first detaching the global fitness function in two, one that represents the *main objective* and other that represents the *secondary objective*, and then by adding an extra penalizing factor that decreases a chromosome's fitness if the fitness of the *main objective* decreases. The intention is to lead the GA (in this phase) in improving the fitness of the *secondary objective* without reducing the fitness of the *main objective*.

If we consider the global fitness function detached in two, f_{glb_1} (eq. 4.6) and f_{glb_2} (eq. 4.7), then, in this phase (and also in phase 2), the global fitness function for a chromosome is given by eq. 4.8, where k_{pnl} is the penalizing factor.

$$f_{glb_1} = f_{V_O} \times f_{I_A} \times f_{I_{DD}} \times f_{I_{SS}} \tag{4.6}$$

$$f_{glb_2} = f_{n_{genes}} \times f_{mos_{area}} \tag{4.7}$$

$$f_{glb} = f_{glb_1} \times f_{glb_2} \times k_{pnl} \tag{4.8}$$

This penalizing factor k_{pnl} is normally 1.0 but is reduced if f_{glb_1} decreases. Some options are implemented in the GA to decide when and how much k_{pnl} should be reduced. The decision about when to reduce k_{pnl} has two options: 1) when f_{glb_1} decreases compared to its value in last iteration's best chromosome; 2) when f_{glb_1} decreases compared to its value in last phase's best chromosome.

The decision regarding how to reduce k_{pnl} also has some options. This factor may be drastically reduced to zero, or may be reduced to another value (between 0.0 and 1.0) without such an absorbing effect. In the latter case, it can have a fixed value or a value that depends on the reduction experienced by f_{glb_1}.

The reduction in k_{pnl} when there is a reduction in f_{glb_1} has a penalizing reinforcing effect on the evaluation of the chromosome, f_{glb}, reducing it more drastically if only the effect of f_{glb_1} were taken into account. The idea is to reduce the probability of selection of this chromosome because it is "loosing" its "quality" regarding the objective evaluated by f_{glb_1}.

In this test, an option was used that sets k_{pnl} to zero if f_{glb_1} is reduced below 0.95 of this phase's best chromosome in the previous iteration. However, it seems (from further testing) that it is preferable to reduce k_{pnl} depending in the reduction experienced by f_{glb_1} when

compared to the fitness in last phases's best chromosome.

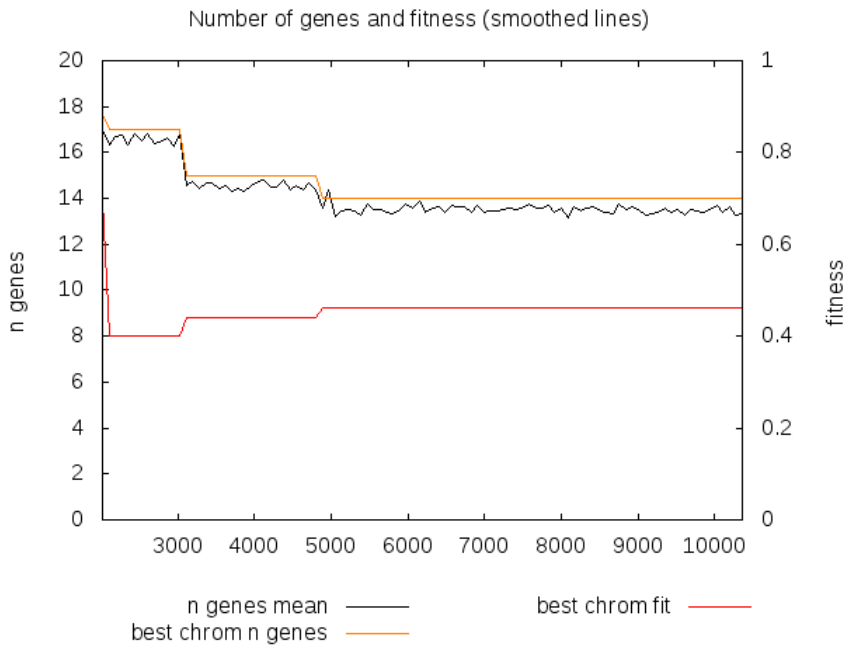


Figure 4.22: Number of genes and best chromosome fitness in phase 1.

Circuit generated in phase 2

Notice that although some changes were made to the circuit in phase 1, namely the removal of some components, most of the transistors remained unchanged in size. The optimization of transistor sizes was left for phase 2, which produced the circuit shown in Figure 4.23.

During phase 2, a few more components were removed (two transistors and a resistor), and almost all the transistors were reduced in size. This was a slow process, taking about 8000 iterations, although most of the time the population was half the size it was in phase 0, which reduced the execution time significantly. As can be seen in Figure 4.24, there were still some adjustments in the number of components until iteration 12 703, but then there were no more, leaving only the optimization of the size of the transistors to be made, which was also a slow process since it took 5678 iterations to be concluded (the end of the phase was determined by fulfillment of a stagnation criterion).

Figures4.25a and 4.25b represent desired and achieved output signals (respectively), and it can be noticed that the achieved output signal has some offset (of $-163\,\text{mV}$) which, similarly to what was done for the previous test, was purposely allowed in the solution of this problem. For the signals shown in Figure 4.25, the circuit obtained in phase 2 also meets the specifications for currents $I_{DD_{rms}}$, $I_{SS_{rms}}$ and $I_{A_{rms}}$, whose values (measured in NGSPICE) are 1.83 mA, 1.73 mA and 1.01 nA, respectively.

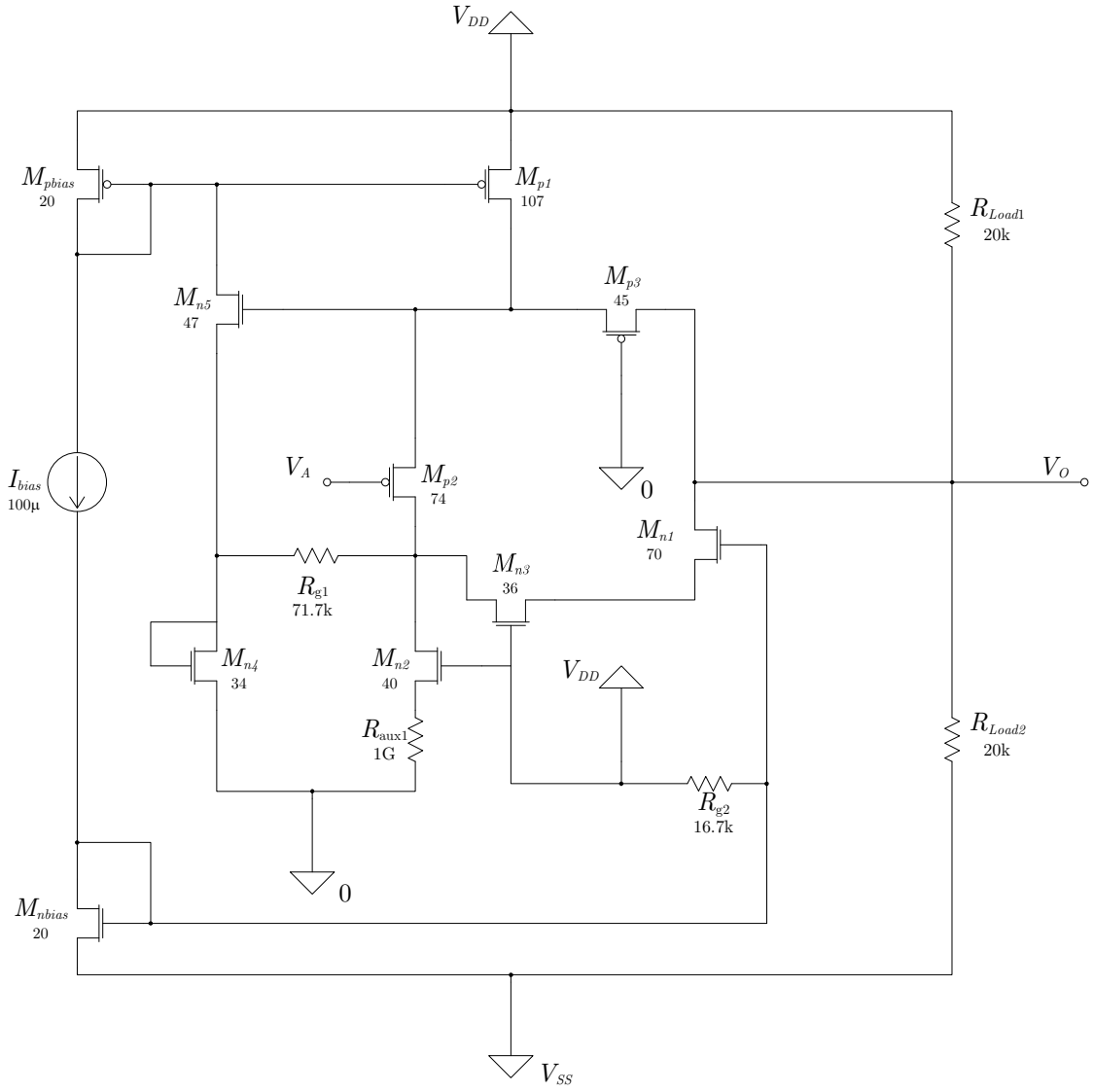


Figure 4.23: DC amplifier with gain 10: circuit generated at the end of phase 2.

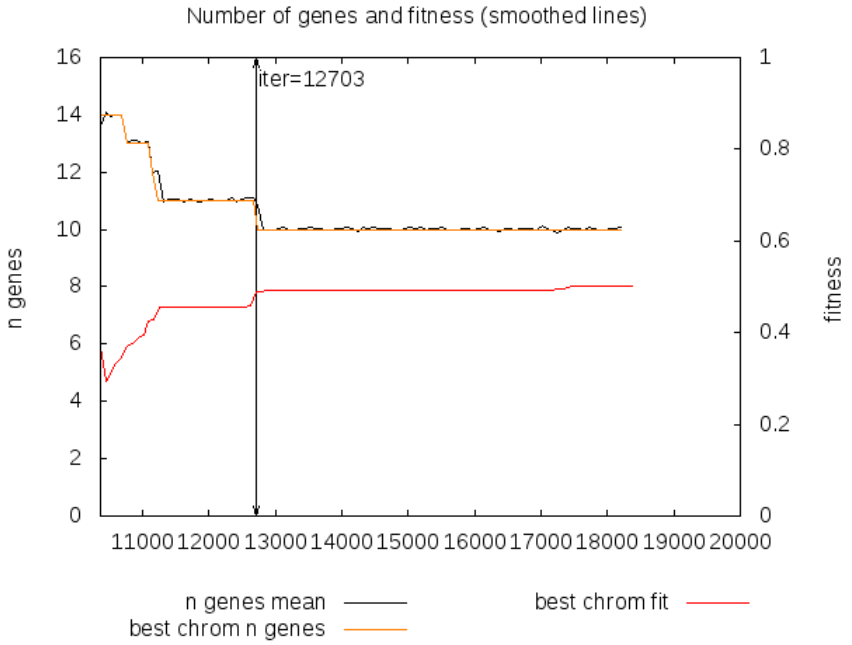
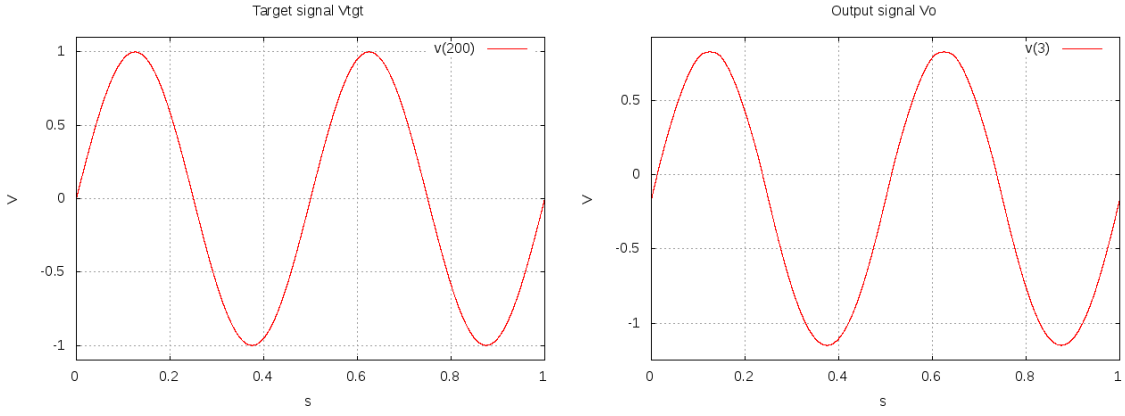


Figure 4.24: Number of genes and best chromosome fitness in phase 2.



(a) Desired (target)

(b) Output

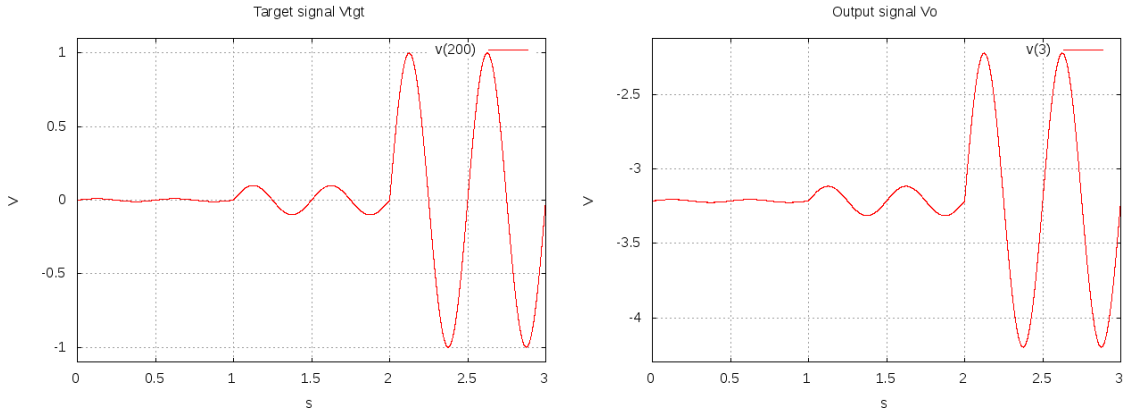
Figure 4.25: Desired (target) and output signals

4.3.2 A 32dB DC Amplifier

This test had a similar goal as the previous one, the synthesis of a DC amplifier using MOSFET transistors, but instead of a gain of 10, the goal in this test was to achieve a voltage gain of 40. All the techniques used in the previous test were also used in this one. However, there were some changes in the circuit parameters and in the embryo circuit.

4.3.2.1 The embryo circuit

The most relevant change in the embryo circuit was the use of a new input signal which generates the desired output signal illustrated in Figure 4.26a. The output signal actually obtained by the circuit synthesized by the GA in this test is shown in Figure 4.26b.



(a) Desired (target) (b) Output

Figure 4.26: Desired (target) and output signals

In this new desired output signal, the amplitude of the sinusoid has three distinct values, each used during two full periods of the sinusoid. The last and higher value is 1 V, which leads to an output swing of 2 V, as intended by the circuit's specifications. The previous value is one order of magnitude lower, and the first value is 0.01 V, *i.e.*, two orders of magnitude lower than the higher value. The intention is to promote a gradual achievement of the final goal by presenting two partial and easier goals to the GA, assuming that it is easier for the GA to synthesize a circuit of gain 40 with an output swing of 0.02 V than it is with an output swing of 2 V. This technique aims to steer the evolution of the circuit by promoting the successive achievement of goals with increasing difficulty.

Another change made in this test was to increase the supply voltages, as shown in Table 4.13. In some preliminary tests done earlier, increasing the supply voltage seemed to facilitate the synthesis of higher gain circuits. However, this needs to be confirmed by a sufficient number of tests where the only variable parameters are V_{DD} and V_{SS} values.

The range of resistor values and the maximum number of nodes were not changed in this test because there was no evidence that the ones used so far were not suitable for the actual purpose.

Table 4.13: Circuit parameters

Positive power supply	V_{DD}	5.0 V
Negative power supply	V_{SS}	−5.0 V
Target signal source gain	k	40
Input signal $v_A(t)$	$A(t)\sin(\omega t)$	$\omega = 2\pi \times 2\,\text{Hz}$ $A(t) = 250\,\mu\text{V}\ ,\ t \in [0\ ,\ 1]\,\text{s}$ $A(t) = 2.50\,\text{mV}\ ,\ t \in [1\ ,\ 2]\,\text{s}$ $A(t) = 25.0\,\text{mV}\ ,\ t \in [2\ ,\ 3]\,\text{s}$
Transistors W/L ratio	W/L	1 to 200
Resistors in evolvable circ	$R_{g1}, R_{g2}..R_{gN}$	10 Ω to 1 MΩ
Max n of nodes in evolvable circ	$node_{max}$	14 (excluding ground)

The range of W/L ratios in MOS transistors was also kept unchanged in this test, although it can be argued that larger ranges should be tested since there is a high occurrence of transistors in parallel in intermediate circuits along the evolutionary process. On the other hand, it is also true that the GA was able to synthesize circuits that fulfilled the objective without using transistors in parallel, or transistors with that ratio close to the upper limit of the allowed range.

Changing some default transistor model parameters

In all results presented so far where MOS transistors were involved, a *LEVEL 1 MOSFET* model was used to simulate such transistors. Although the use of other models was attempted in several experimental runs of the GA, the time penalty for using higher level models, such as *BSIM3*, was found to be extraordinarily high. This dictated that the *LEVEL 1 MOSFET* model should be the default model used by the GA, given the computing power available for this research work. In this context, "default" means that all tests should run at this level unless there is a good reason to increase it, and expect a corresponding increase in test execution time.

However, the default parameters of the *LEVEL 1 MOSFET* model use the same value for the process transconductance for both NMOS and PMOS transistors, which often produces unrealistic results, even at the operating point of simple circuits such as the embryo circuit used in this test. In order to obtain closer results for the operating point of the embryo circuit when comparing simulations performed with the *LEVEL 1 MOSFET* model and the *LEVEL 8 version=3.3.0 MOSFET* model (the *LEVEL 8 version=3.3.0 MOSFET* model is a version of the *BSIM3* model in NGSPICE), some default parameter values of the *LEVEL 1 MOSFET* model have been changed and are now set in the embryo netlist for all circuit simulations performed by the GA. An example of such a netlist is shown in Figure 4.27, which, in addition to providing a global view of the netlist, also allows for checking that the default values of `Kp`, `Vto`, and `LAMBDA` parameters have been superseded by new values.

```
Amplifier with MOSFETs and VLC
*
* Power supply
VDD     7   0   5.0V
Rvdd    7   0   1G
VSS     8   0   -5.0V
Rvss    8   0   1G
*
* Simul parameters
.PARAM n_width=200u                    $ Width of NMOS  transistors used in bias network.
.PARAM n_len=10u                       $ Length of NMOS transistor used in bias network.
.PARAM pn_ratio=2.7                    $ Heuristic ratio between PMOS and NMOS process
transconductance.
*
.PARAM p_width={n_width * pn_ratio }    $ Width of PMOS  transistors used in bias network.
The W/L ration
                                       $ is such that Vgs will be aprox. simetrical for the
                                       transistors of the bias network.
.PARAM p_len={n_len}                    $ Length of PMOS transistor used in bias network.
*
.PARAM kp_nmos=2.07E-05                 $ Kp=2.07E-05 is the default Kp used in Level 1 (for
both NMOS and PMOS).
                                       $ Use the same Kp for NMOS in this run,
.PARAM kp_pmos={kp_nmos / pn_ratio}     $ but use a lower Kp for PMOS transistors, which is
more reallistic (and closer to BSIM3).
*
.PARAM freq={2}                        $ Test frequency
.PARAM gain={40.0}                     $ Desired gain of the amplifier.
.PARAM Vo_amp={1}                      $ Desired output amplitude.
*
.PARAM Vi2_amp={Vo_amp / gain / 100}    $ Lower input amplitude.
.PARAM Vi1_amp={Vo_amp / gain / 10}     $ Intermidiate input amplitude.
.PARAM Vi_amp={Vo_amp / gain}           $ Higher input amplitude.
*
.PARAM Va2_amp={Vi2_amp}                    $ Lower input amplitude voltage source parameter.
.PARAM Va1_amp={Vi1_amp - Va2_amp}          $ Intermidiate input amplitude voltage source
parameter.
.PARAM Va_amp={Vi_amp - Va1_amp - Va2_amp}  $ Higher input amplitude voltage source parameter.
*
* Bias network
Mpbias  2   2   7       7   MY_PMOS  W={p_width} L={p_len}
Mnbias  4   4   8       8   MY_NMOS  W={n_width} L={n_len}
Ibias   2   4   100u
*
* Load resistors
Rload1      7       3   20k
Rload2      3       8   20k
*
* Input sources
Va          1       300     DC  0   SIN (0 {Va_amp}   {freq} 2)
Va1         300     301     DC  0   SIN (0 {Va1_amp}  {freq} 1)
Va2         301     0       DC  0   SIN (0 {Va2_amp}  {freq} 0)
Rs1         1       0       1G
*
* Target source
Etgt        200     0       1       0       {gain}
Rtgt        200     0       1G
*
* Pseudo noise sources
VNoise1     100     99      DC  0   SIN (0 1.0 {1000*freq} 0)
VNoise2     99      0       DC  0   SIN (0 1.0 {100*freq}  0)
RNoise      100     0       1G
*
* Transistor models
.MODEL MY_PMOS PMOS level=1 Kp={kp_pmos}    Vto=-1.6    LAMBDA=0.04
.MODEL MY_NMOS NMOS level=1 Kp={kp_nmos}    Vto=1.6     LAMBDA=0.04
*
* Analysis
.TRAN 300u 3
*
.END
```

Figure 4.27: Spice netlist for the embryo circuit.

With these new values for those parameters, the operating point analysis of some simple circuits (such as the embryo circuit of this test) did indeed show closer values when comparing simulations with the *LEVEL 1 MOSFET* model and the *LEVEL 8 version=3.3.0 MOSFET* model. However, this does not guarantee any degree of similarity of results for other analyses.

4.3.2.2 Synthesized circuits at the end of each phase

The circuit of Figure 4.28 was obtained at the end of phase 0 of this run, after 84 iterations of the GA. A detail of the fitness evolution for the first 160 iterations is shown in Figure 4.29,

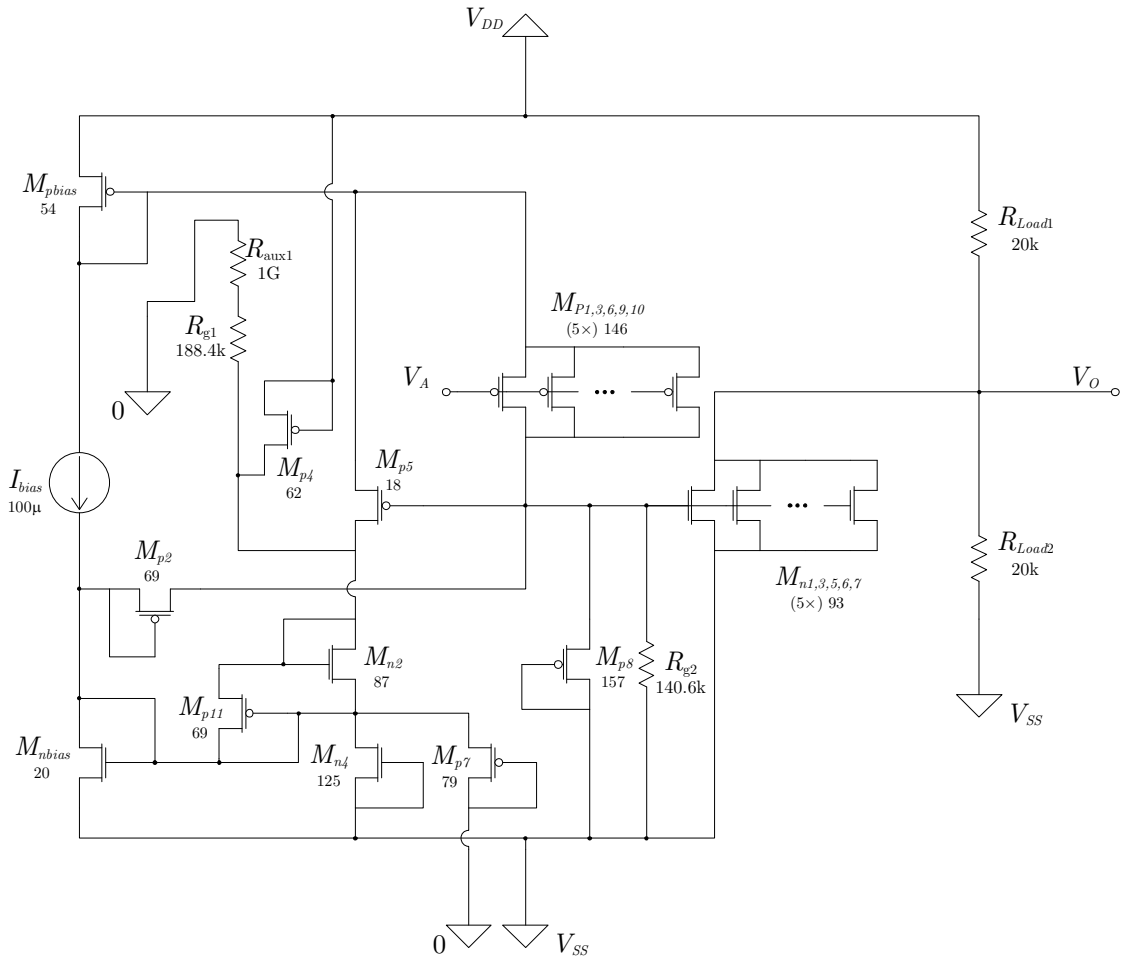


Figure 4.28: DC amplifier with gain 40: circuit generated at the end of phase 0.

where it can be seen that the end of this phase was not decided by stagnation, but by the best chromosome in the population reaching a fitness threshold of 0.95.

This is a variant to the methodology of waiting for stagnation that can be used in phase 0 if the overall parametrization of the GA is sufficiently fine-tuned so that a given fitness threshold has a sufficiently known meaning of success. Of course, there is a risk that, by stopping evolution only when a fitness threshold is reached, we may not get the best circuit that could possibly be obtained at this phase, but it should be remembered that this phase

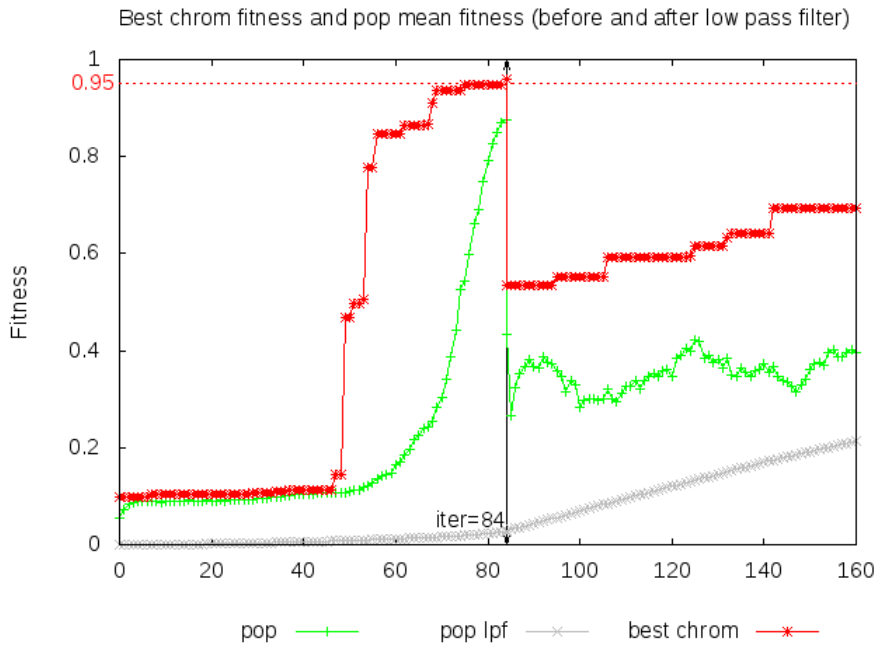


Figure 4.29: Fitness values until iteration 160.

is followed by other phases that are dedicated to circuit optimization.

The circuit of Figure 4.28 has several redundant or useless components that were removed during phase 1, which produced the circuit shown in Figure 4.30. This phase (phase 1) lasted until iteration 2081 and was terminated because of fitness stagnation, and its fitness evolution can be seen in Figure 4.32. In the circuit of Figure 4.30 it can be seen that transistors M_{p1} and M_{n1} are the largest ones, having W/L ratios near the maximum allowed for this test (which was 200).

The next phase was dedicated to the optimization of the MOS area used by transistors, and resulted in the circuit shown in Figure 4.31 (some operating point measurements have been added to this schematic). During this phase, all transistors suffered some reduction in their W/L ratio, which means that the total area was reduced (note that, in all tests, the length L of the transistors is kept constant throughout the run). In particular, notice how the W/L ratios of transistors M_{p1} and M_{n1} have been reduced from 190 to 67 and 81, respectively.

The output signal obtained by this circuit has already been shown in Figure 4.26b, where an offset voltage of $-3.34\,\mathrm{V}$ is observed. It should be noted that (similarly to what was done in previous tests) it was not used any form of penalty for the existence of this offset voltage, so the GA was free to find the best solution without taking this offset into account.

For a continuous sinewave of 1 V amplitude at its output, the circuit obtained in phase 2 meets the specifications for currents $I_{DD_{rms}}$, $I_{SS_{rms}}$ and $I_{A_{rms}}$, whose values are 0.81 mA, 0.81 mA and 1.00 nA, respectively.

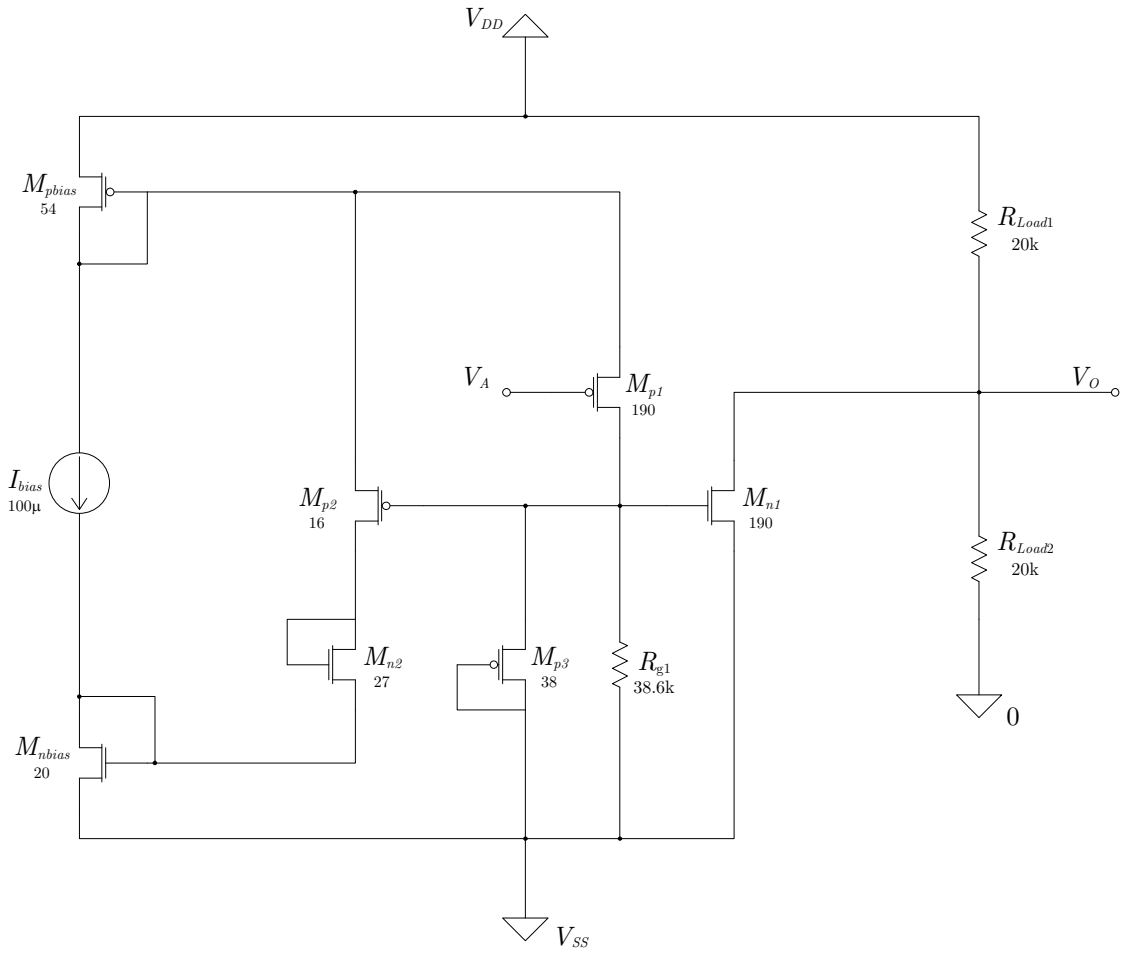


Figure 4.30: DC amplifier with gain 40: circuit generated at the end of phase 1.

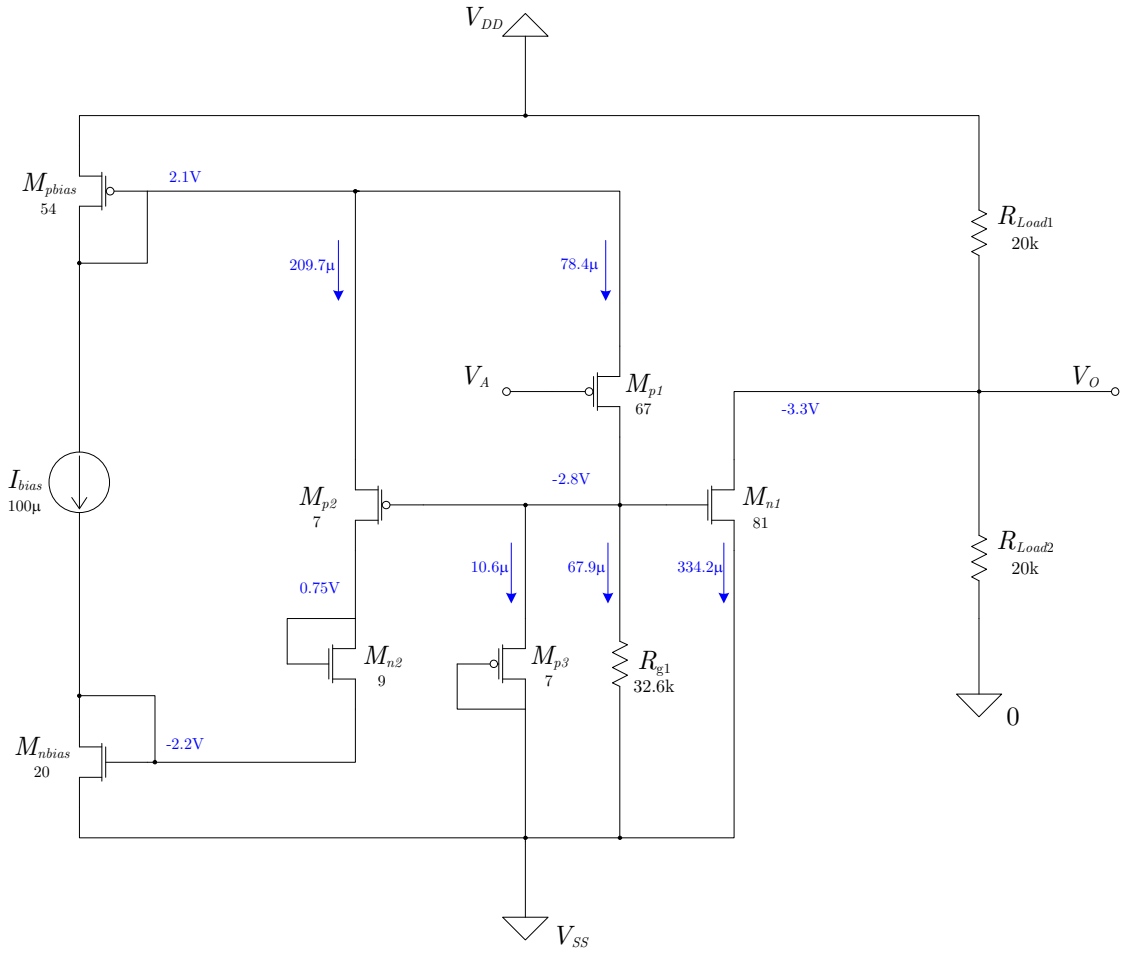


Figure 4.31: DC amplifier with gain 40: circuit generated at the end of phase 2.

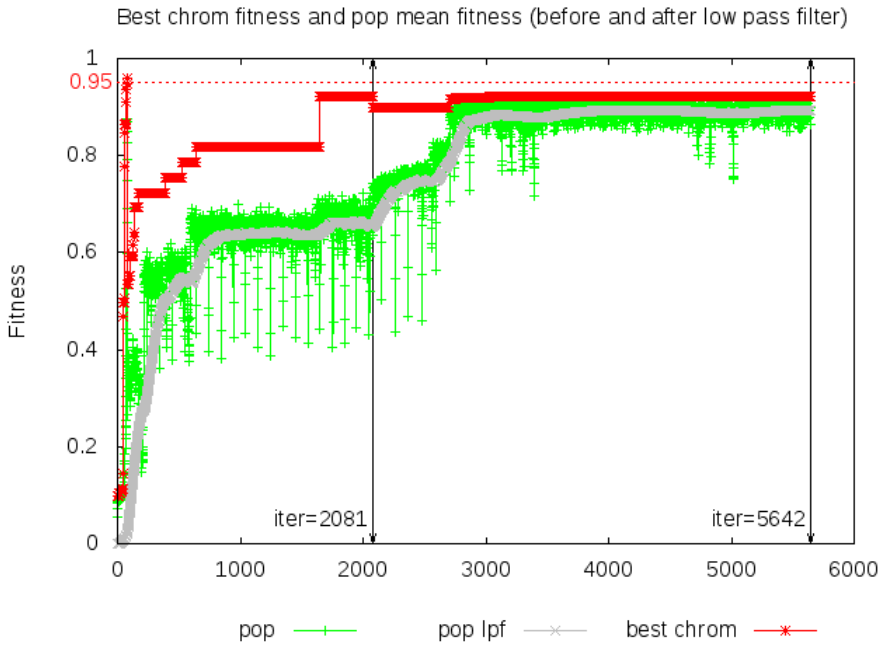


Figure 4.32: Fitness values until iteration 5642.

4.3.2.3 Genes and fitness across phases

As in the previous test, this test also shows a significant relationship between the number of genes in chromosomes and the evolution of fitness, as illustrated in the next plots (Figures 4.33, 4.34, 4.35). This relationship is particularly visible in the best chromosome in the population when examined over the course of evolution, and is an important aspect of these tests made possible by the use of VLC.

Genes and fitness across phase 0

Figure 4.33 shows the evolution of the number of genes of the best chromosome along with its fitness, as well as the average number of genes used by each chromosome for phase 0 of this test. The end of this phase occurred at iteration 84 (marked with a vertical arrow in the graph), when the fitness value reached a predefined threshold of 0.95 (which was the stopping criterion adopted for this phase).

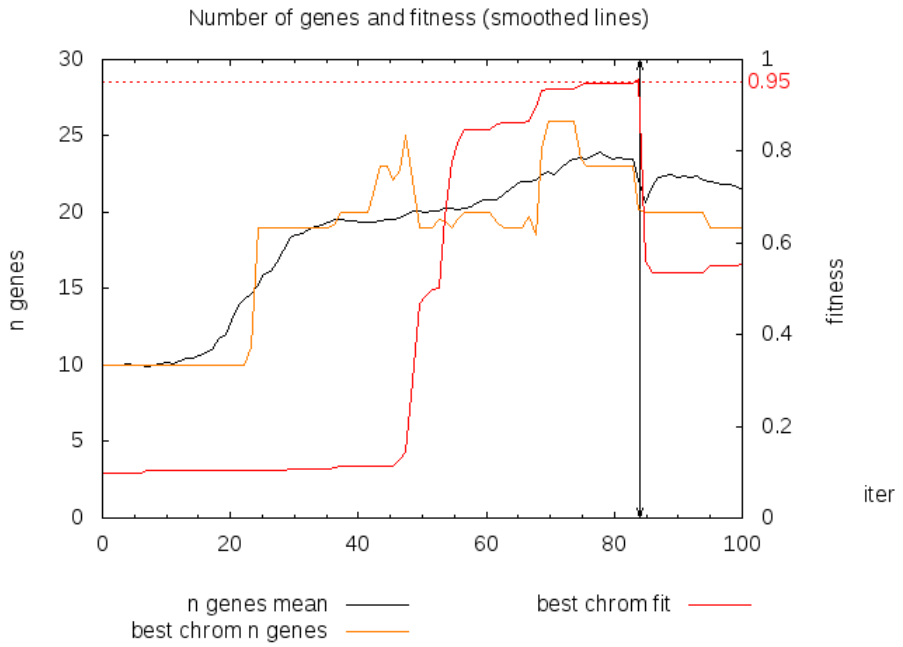


Figure 4.33: Number of genes and best chromosome fitness in phase 0.

To try to identify what is happening during this phase, it can be broken down into several parts. In a first part, considered since the beginning until about iteration 48, the average number of genes in the chromosomes grows consistently for almost all iterations, although the best chromosome fitness remains at a low value of about 0.1. The number of genes of the best chromosome suffers a sudden increase from 10 to 19 at about iteration 24, but with no evidence of any impact on its fitness.

At about iteration 43, the number of genes of the best chromosome starts to increase (non-monotonically, up to a maximum of 25) leading to the first significant increase in its fitness, a stepwise increase from about 0.1 to about 0.5, which occurs between iterations 48 and

53. However, by the time this fitness value is reached, the number of genes has already returned to 19. It seems that this increase in the number of genes was necessary for the occurrence of that increase in fitness, even though these extra genes are no longer needed later. This pattern of behavior (where an increase in the number of genes immediately precedes an increase in fitness) is observable in numerous tests similar to this one.

From about iteration 54 to 61 there is another stepwise increase in the fitness of the best chromosome, which was not accompanied or preceded by any relevant variation in the number of genes on that chromosome (although the average number of genes continued its upward trend).

Starting with iteration 67, there is another stepwise increase in the fitness of the best chromosome, accompanied by an increase in its number of genes, which rises from about 19 to 26. After this increase in fitness, reaching the threshold of 0.95 is approached, but before it is reached, the number of genes decreases from 26 to 23, which seems to be the principle of the "cleaning" effect mentioned before. And when, in iteration 84, that threshold is finally reached, the number of genes has already decreased to 20, which undoubtedly results from that "cleaning" effect on the circuit.

Genes and fitness across phase 1

Phase 1 lasted until iteration 2081 and, as shown in Figure 4.34, the number of genes decreased in a sustained manner during this phase.

The fitness of the best chromosome had its usual sudden decrease when the phase changed from 0 to 1, caused by the shift of weights, and then increased in a sustained way until the phase stopped (by stagnation). This behavior, both for the evolution of the number of genes and for the evolution of the fitness, seems to be a recurrent pattern for this phase.

Genes and fitness across phase 2

Phase 2, which lasted until iteration 5642 and was dedicated to optimizing the MOS area used by transistors in the circuit, did not change the number of genes in the best chromosome, since no component was removed from the circuit, as shown in Figure 4.35. The average number of genes in the chromosomes converged to the same value of the best chromosome, which is consistent with what happens during most of this phase: local exploration in search of the best sizes of existing transistors.

4.3.3 A 38dB DC Amplifier

As in the previous test, in this one the goal was the synthesis of a DC amplifier using MOSFET transistors, but this time with a gain twice the gain of the last synthesized amplifier, *i.e.*, a gain of 80. All the techniques used in the previous test were also used in this one. In this test, however, the possibility that the circuit could contain capacitors was added and a single pole low pass frequency response was included in the desired specifications. An extra phase, dedicated to the optimization of the frequency response of

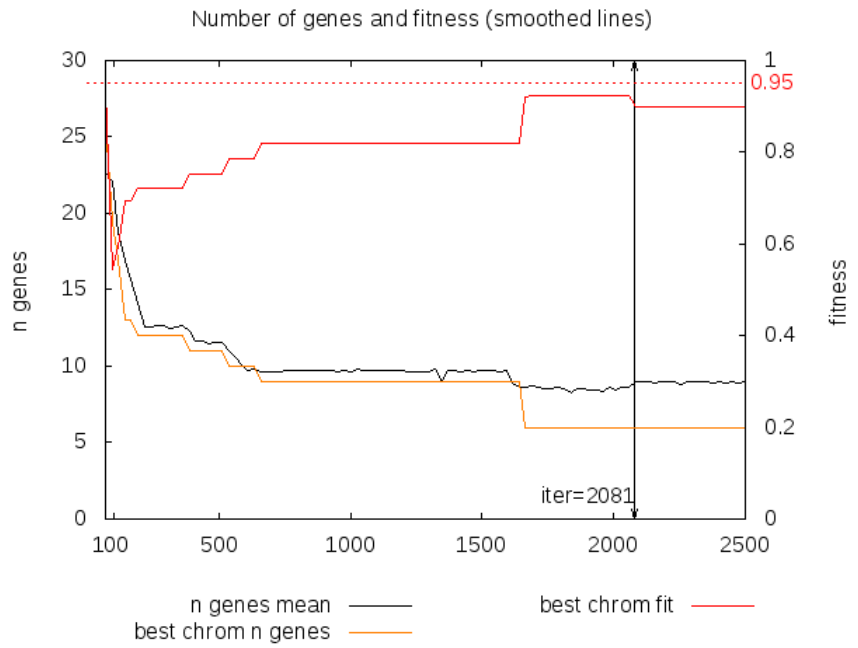


Figure 4.34: Number of genes and best chromosome fitness in phase 1.

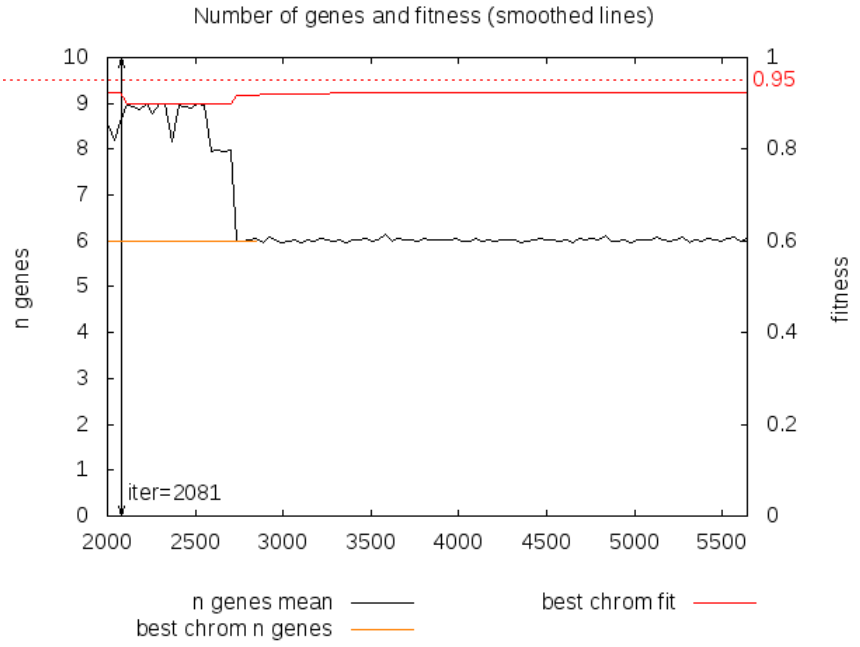


Figure 4.35: Number of genes and best chromosome fitness in phase 2.

the circuit, was added, so the evolution in this test consists of 4 phases.

Also, a term has been added to the global fitness function that slightly penalizes the existence of offset at the output of the amplifier. This was done in an attempt to minimize this offset, but in preliminary runs it was verified that this penalizing factor can be highly disruptive to the evolution of the GA, so its weight in the global fitness function was kept low.

4.3.3.1 Embryo circuit and frequency response

The embryo circuit used in this test is very similar to the one used in the previous test, as shown in Figure 4.36. However, in this embryo circuit, the target signal V_{tgt} is the low-pass filtered version of the signal produced by source kV_A (with $k = 80$ in this test). The low-pass filtering is produced by R_1 and C_1, which were dimensioned so that the pole frequency is 100 kHz. Hence, it was expected that the GA produces an amplifier with a bandwidth of 100 kHz.

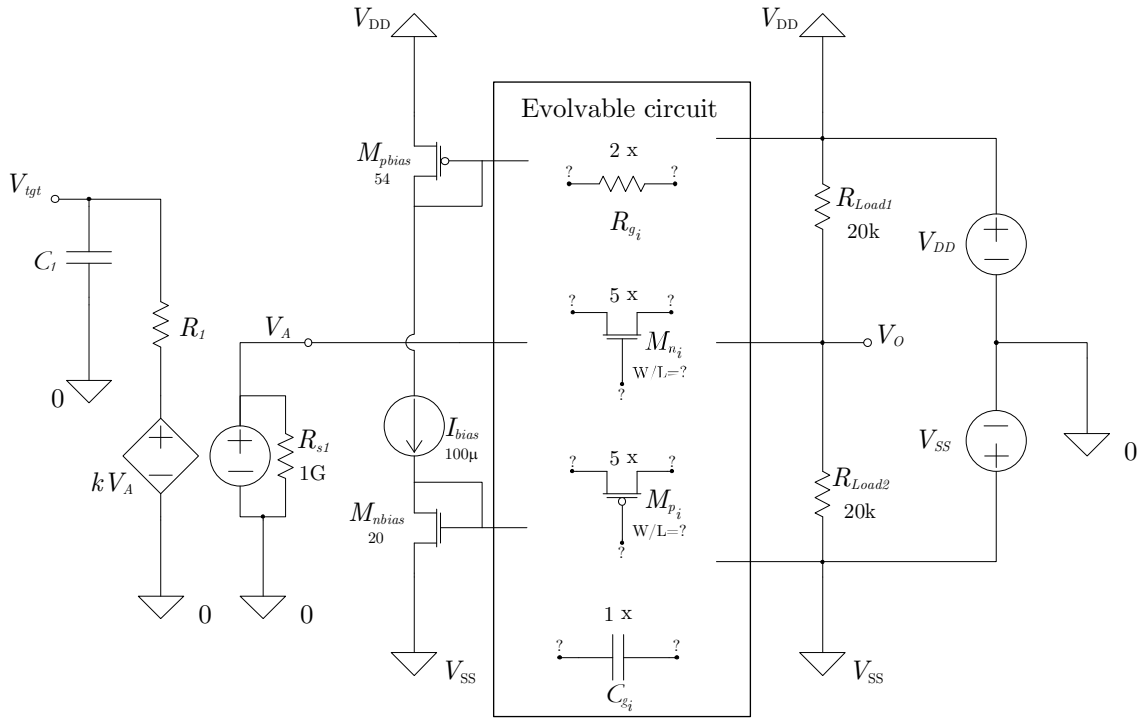


Figure 4.36: Embryo circuit for a DC amplifier with gain 80 using MOS transistors.

To achieve this goal of sizing the frequency response of the amplifier, an AC analysis is performed (in addition to the TRAN analysis already carried out) and a merit function is used that measures the difference between the desired frequency response and the actual one. Also, the GA is allowed to use capacitors in the synthesis of the circuit. However, the existence of more than one capacitor in the circuit is heavily penalized, so it is expected that the GA manages to match the desired frequency response with only one capacitor in the circuit. Note that there is already a capacitor in the embryo circuit, although this is not

mandatory because in this test the mutation operator can add capacitors to the circuit at any stage of the evolution.

Another change made in this test was to increase the upper limit of the allowed W/L ratio range in transistors from 200 to 500, as shown in Table 4.14. The intent of raising this upper limit was to facilitate the synthesis of higher gain circuits, keeping in mind that the fitness function includes penalization for high MOS areas used by transistors. Thus, although this upper limit is now much higher than in previous tests, it was hoped that intermediate circuits would use such large W/L ratios if necessary, but that final synthesized circuits would not.

Table 4.14: Circuit parameters

Positive power supply	V_{DD}	5.0 V
Negative power supply	V_{SS}	−5.0 V
Target signal source gain	k	80
Input signal $v_A(t)$ for TRAN analysis	$A(t)\sin(\omega t)$	$\omega = 2\pi \times 2\,\text{Hz}$ $A(t) = 250\,\mu\text{V}\ ,\ t \in [0\ ,\ 1]\,\text{s}$ $A(t) = 2.50\,\text{mV}\ ,\ t \in [1\ ,\ 2]\,\text{s}$ $A(t) = 25.0\,\text{mV}\ ,\ t \in [2\ ,\ 3]\,\text{s}$
Input signal $v_A(t)$ for AC analysis	$A\sin(\omega t)$	$A = 1.0\,\text{V}$ Freq. log sweep from 1 Hz to 1 MHz
Transistors W/L ratio	W/L	1 to 500
Resistors in evolvable circ	$R_{g1}, R_{g2}..R_{gN}$	10 Ω to 1 MΩ
Max n of nodes in evolvable circ	$node_{max}$	14 (excluding ground)

The maximum number of nodes and the range of resistor values were left unchanged, as there was no evidence (from previous tests) that there would be any advantage in changing them.

4.3.3.2 Synthesized circuits at the end of each phase

The circuit shown in Figure 4.37 was obtained at the end of phase 0 of this run, after 1919 iterations of the GA. Fitness evolution during this phase can be seen in Figure 4.38 (in the first 1919 iterations). As in the previous test, the stopping criterion for this phase was not stagnation, but the achievement of a fitness threshold of 0.95 by the best chromosome in the population. The use of this stopping criterion usually shortens the duration of this phase at the possible cost of not obtaining a better circuit, namely in terms of the number of components used, and in fact this circuit still has some redundant or useless components. However, it is expected that the following phases will lead to a further and satisfactory optimization of this circuit.

The next phase produced the circuit shown in Figure 4.39, after 3424 iterations of the GA.

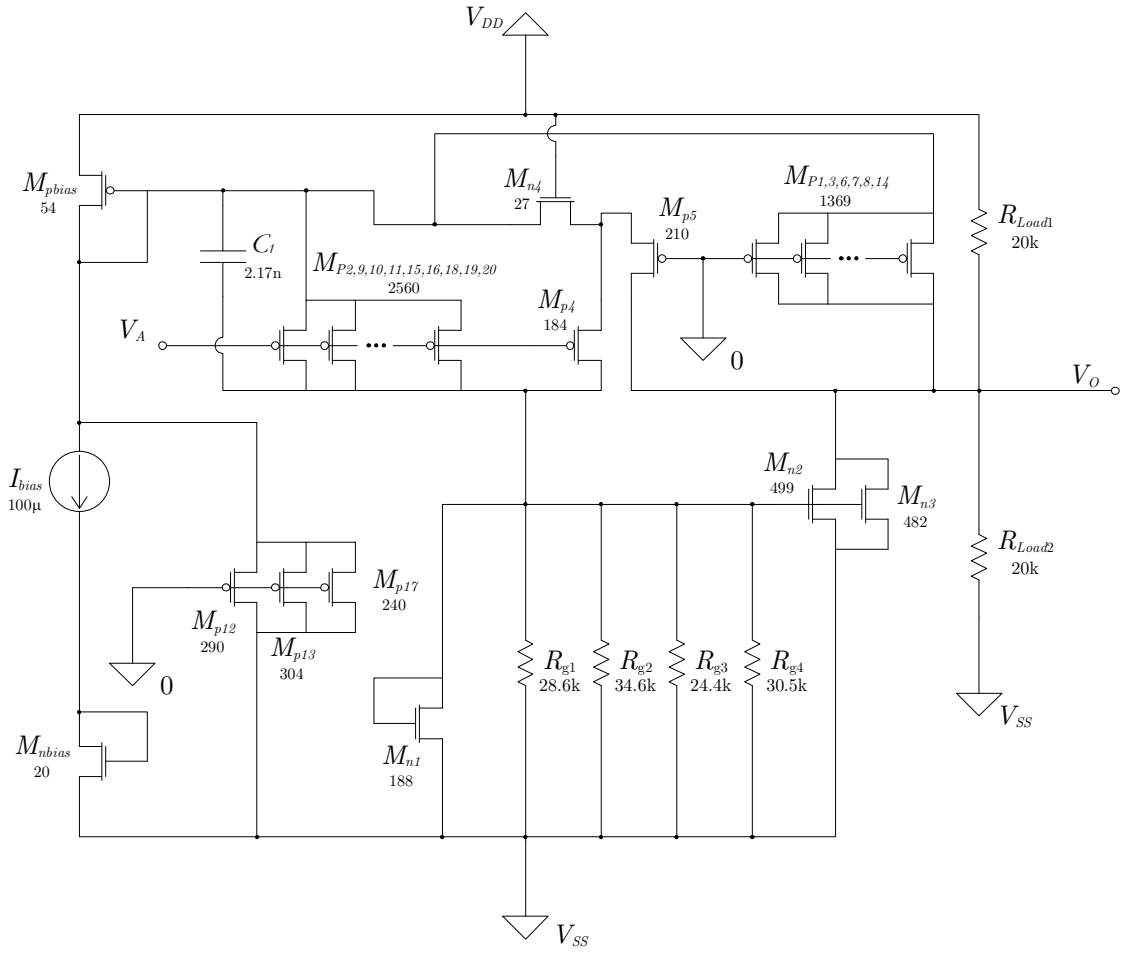


Figure 4.37: DC amplifier with gain 80: circuit generated at the end of phase 0.

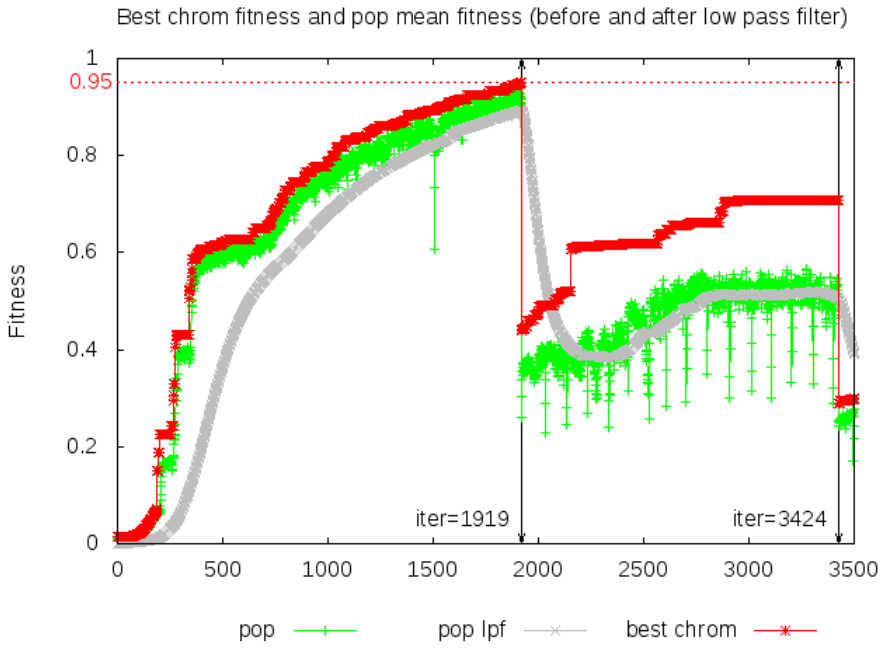


Figure 4.38: Fitness values until iteration 3500.

Fitness evolution for this phase can be seen in Figure 4.38, between iterations 1920 and 3424.

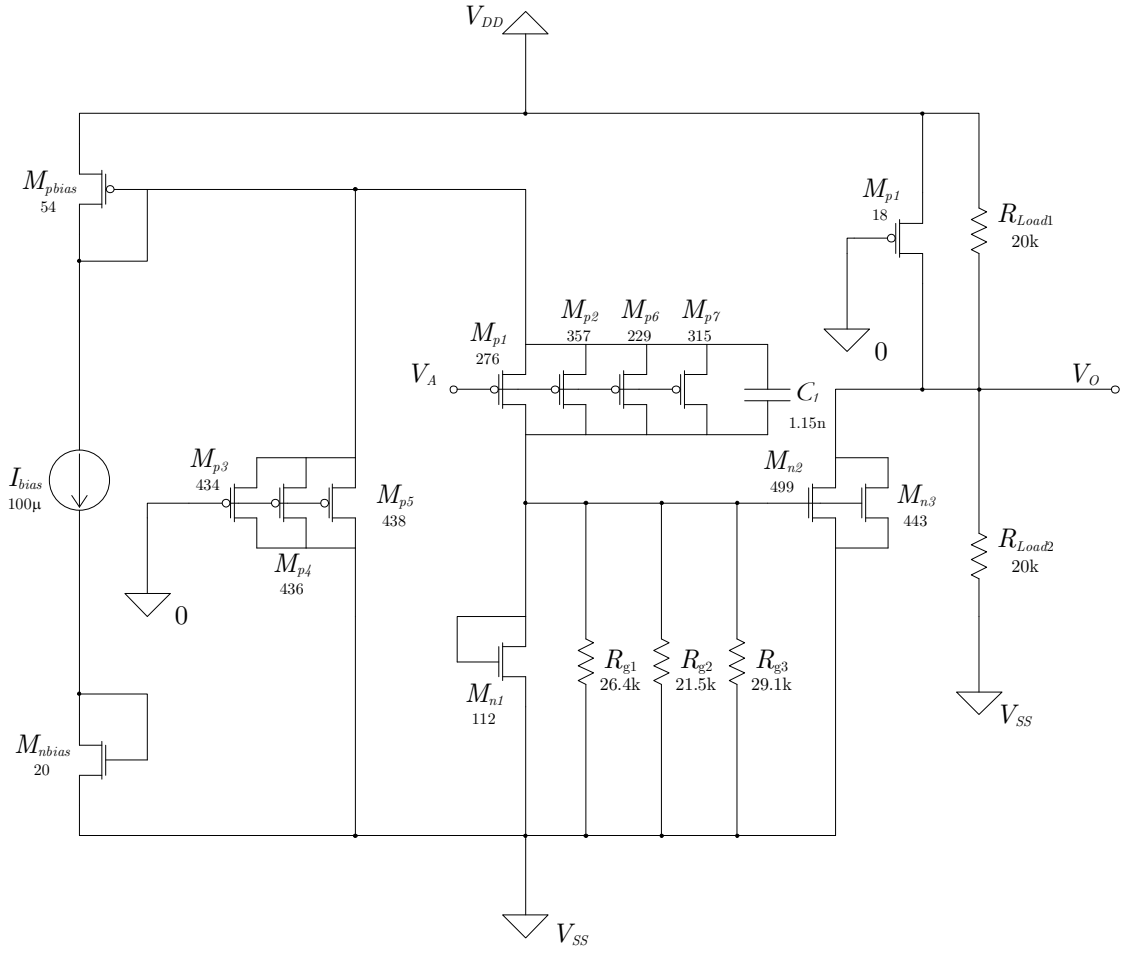


Figure 4.39: DC amplifier with gain 80: circuit generated at the end of phase 1.

It is worth noting that although this circuit has already been somewhat "cleaned" during phase 1, and some useless components have been removed, there are still some redundant components, namely resistors R_{g1}, R_{g2}, and R_{g3}.

As in the circuit obtained in phase 0, this circuit also has one capacitor, which is responsible for the dominant pole of the amplifier. By this time, at the end of phase 1, it may not yet guarantee the desired frequency response, but it is already positioned in the same place on the circuit where it will remain until the final circuit is synthesized (as will be seen later).

As in the previous test, phase 2 was devoted to optimizing the MOS area used by the circuit's transistors, resulting in the circuit shown in Figure 4.40. The criterion to stop this phase was stagnation, and it resulted in a rather long phase, as it lasted until iteration 104 879. Fitness evolution for this phase is shown in Figure 4.41.

During this phase, the GA tried to minimize the total MOS area used by the circuit's transistors without compromising other specifications, in particular the similarity between

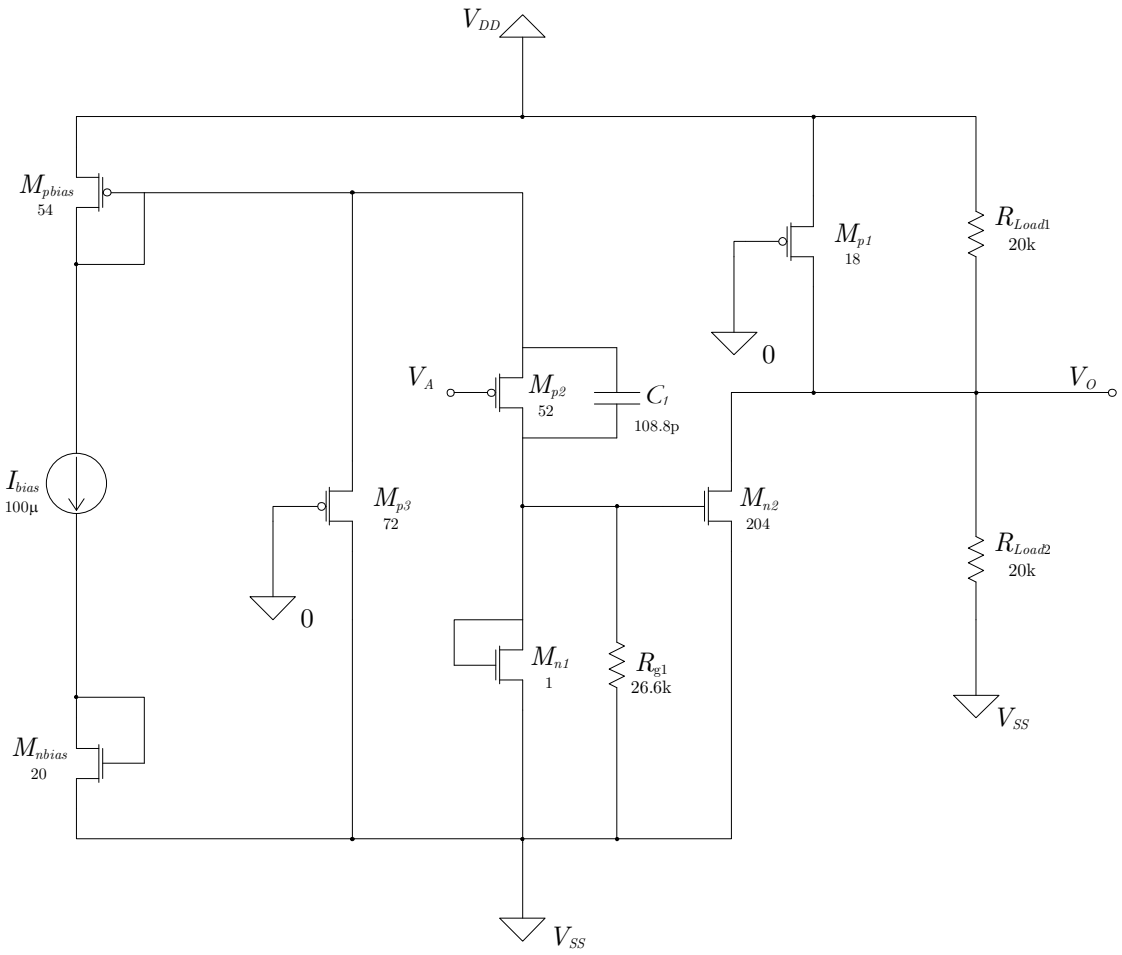


Figure 4.40: DC amplifier with gain 80: circuit generated at the end of phase 2.

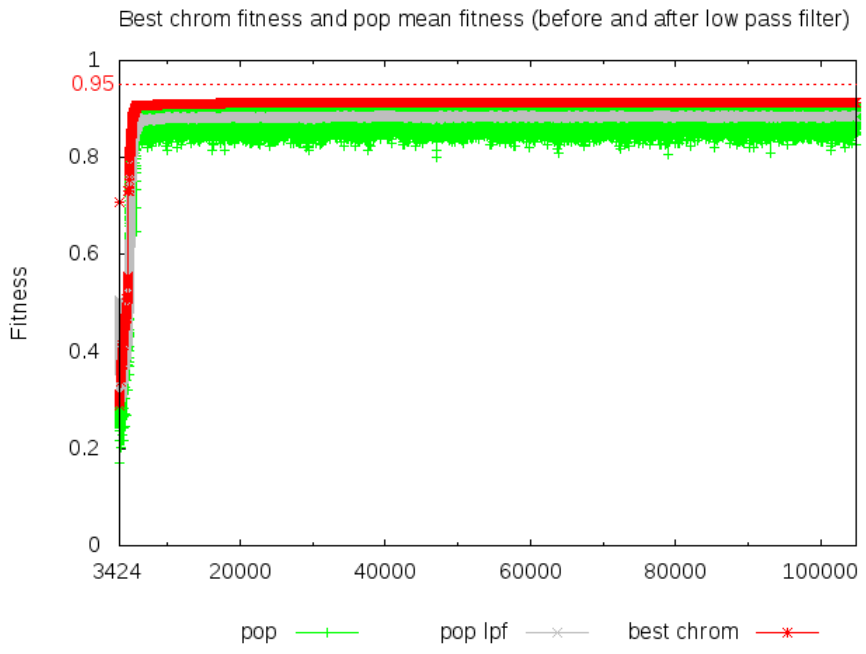


Figure 4.41: Fitness values in phase 2.

the desired and actual V_O signals, which turned out to be a difficult task, as evidenced by the time taken by the GA. In fact, the GA succeeded not only in reducing the area occupied by transistors, but also in reducing the number of components and in sizing capacitor C_{g1} to a value very close to its final value in this test, as will be seen below.

The graphical representation of fitness evolution for this phase, as shown in Figure 4.41, can shed little light on what is happening within the GA, as it can give the impression that nothing is happening between (approximately) iterations 10 000 and 100 000. However, during this period, the GA is searching for solutions that increase the global fitness function (as usual), but without decreasing a partial fitness function that is considered the primary (or mandatory) objective in this problem, similarly to what was described by equations 4.6, 4.7, and 4.8 regarding the test presented in Section 4.3.1. The stopping criterion for this phase was stagnation, so if the GA continued to search for a better solution until iteration 104 879, it means that up to that point there was occasionally enough increase in either the fitness of the best chromosome or in the average fitness of the population to reset the counters that ultimately denote the stagnation state (which happens if these counters are not reset to some iteration limit). Of course, this may also mean that the parameters used in the stagnation decision need some fine-tuning. After all, they are simply heuristic parameters that result from experience acquired along this work.

In this test, a fourth phase was added that was dedicated to the optimization of the frequency response of the amplifier. This turned out to be a much shorter phase than the previous one. Phase 3 stopped at iteration 105 831, producing the circuit shown in Figure 4.42, to which some operating point measurements were added.

During this last phase, only minor changes were made to the circuit. Fitness evolution for this phase is shown in Figure 4.43. From this figure, it is difficult to see what happened to the fitness values during this phase, and one might even erroneously conclude that nothing happened. However, looking more closely at the fitness data produced during this phase, it can be verified that the initial fitness of the best chromosome in this phase is $8.953\,360 \times 10^{-1}$, but at the end of the phase it is $9.047\,160 \times 10^{-1}$. It was the adjustment of the values of some components of the circuit corresponding to this small variation in the fitness that occurred during this phase.

In this circuit, the largest transistor has a W/L ratio of 207. As mentioned before, although the maximum W/L ratio allowed in this test was 500, and although some transistors in intermediate circuits do in fact use W/L ratios close to this maximum value, no transistor in the final circuit has a W/L ratio close to this maximum. Even though this behavior could not be guaranteed, it was in fact the desired behavior of the GA, and this outcome contributes to the belief that the existence of a maximum limit of this order facilitates the synthesis of this type of circuit (although this needs to be confirmed by further testing).

The offset voltage at the output of the amplifier is not zero, as desired, but is approximately $-0.95\,\mathrm{V}$. This is a better value than the one obtained in the previous test (described in

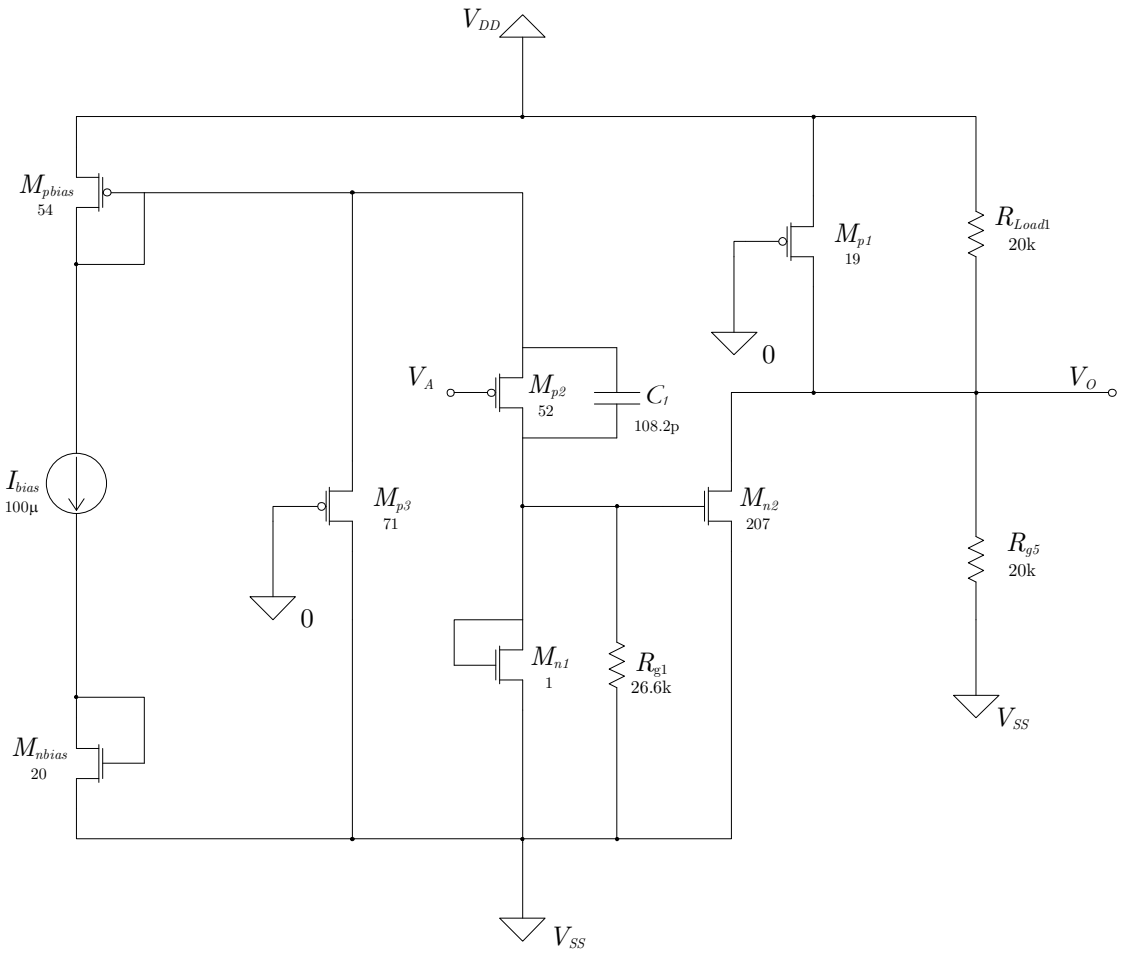


Figure 4.42: DC amplifier with gain 80: circuit generated at the end of phase 3.

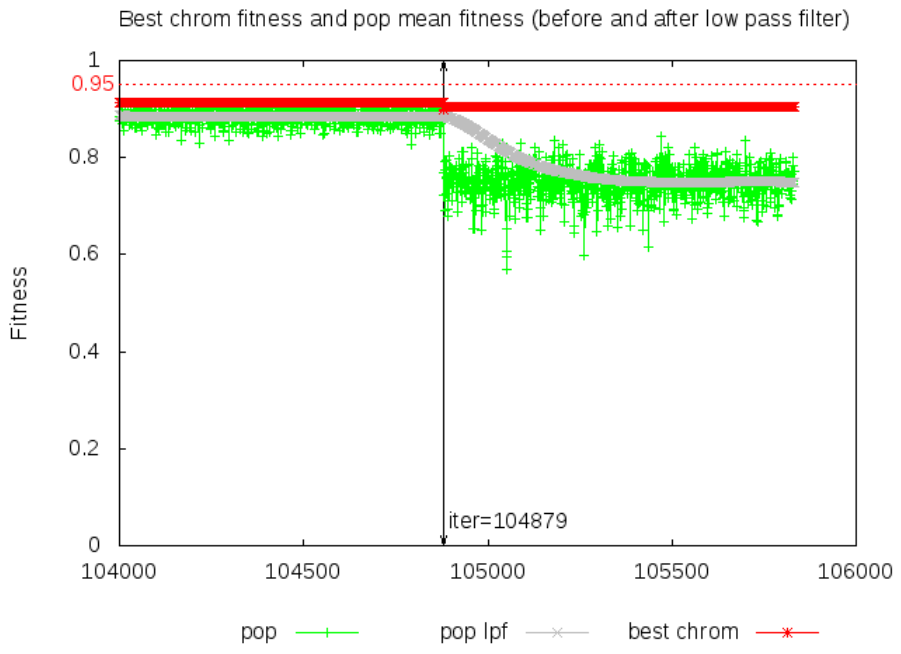
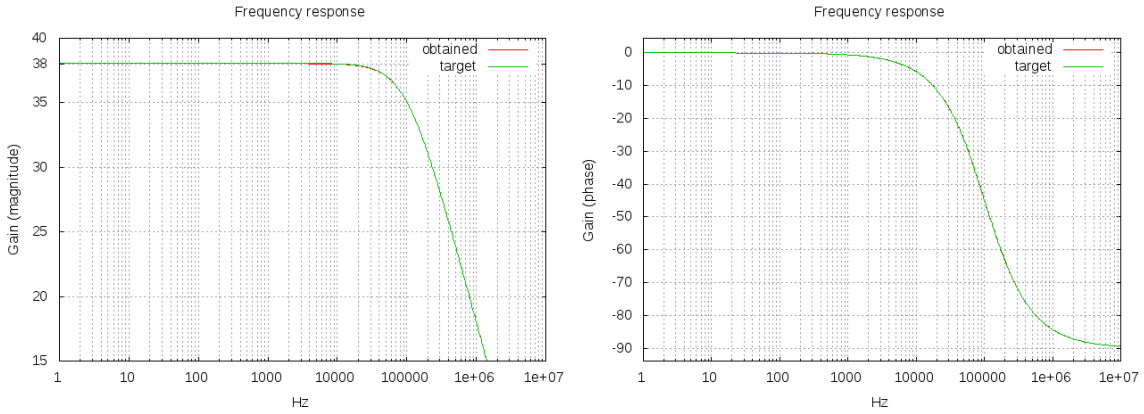


Figure 4.43: Fitness values in phase 3.

Section 4.3.2), which was approximately −3.34 V, and obtaining it is possibly related to the use of the function that penalizes the existence of a non-zero offset. However, without further testing focused on this subject, it is not possible to know how much influence that function actually has in reducing the absolute value of the offset voltage.

This phase was dedicated to the optimization of the frequency response of the amplifier, which was not a specification in previous tests. The frequency response plots for the circuit obtained in this phase are shown in Figure 4.44. It can be seen in both plots, either in the magnitude response plot and in the phase response plot, that the frequency response obtained is very close to the desired one, since the respective traces overlap in most of the frequency range.



(a) Gain magnitude plot.

(b) Gain phase plot

Figure 4.44: Frequency response at phase 3

This result shows that the GA was able to select a location in the circuit for a single capacitor and size it correctly to create a dominant pole at the intended frequency. However, when comparing the frequency response obtained in this phase with those obtained in the previous phases, it appears that there are no major differences, suggesting that some changes in the parameterization of the phases might be appropriate.

One possible change could be to simply eliminate this last phase, while keeping the parameterization of the previous ones. However, following what has been done with the introduction of phases in the evolution of the GA, dedicating each phase to the optimization of a specific characteristic of the circuit, it would perhaps be preferable to further reduce the weight of the function that deals with the frequency response in the first phases (which was already a relatively low weight), in order to eventually reduce the difficulty of the tasks of these phases and, consequently, to see the duration of these phases reduced.

As in the previous test, currents $I_{DD_{rms}}$, $I_{SS_{rms}}$ and $I_{A_{rms}}$ were measured (for a continuous sine wave of 1 V amplitude at the output of the circuit generated in this phase), and the values obtained were 1.65 mA, 1.65 mA and 1.00 nA, respectively. These values meet the specifications for these currents.

4.3.3.3 *Genes and fitness across phases*

Once again, as in the previous tests, there is an important relationship between the number of genes in chromosomes and the evolution of fitness within each phase, as will be discussed in the following paragraphs.

Genes and fitness across phase 0

Figure 4.45 shows the evolution of the number of genes of the best chromosome along with its fitness, as well as the average number of genes used in the chromosomes for phase 0 of this test. The end of this phase occurred at iteration 1919 (marked on the graph with a vertical arrow), when the fitness value reached a predefined threshold of 0.95, which, as in the previous test, was the stopping criterion adopted for this phase.

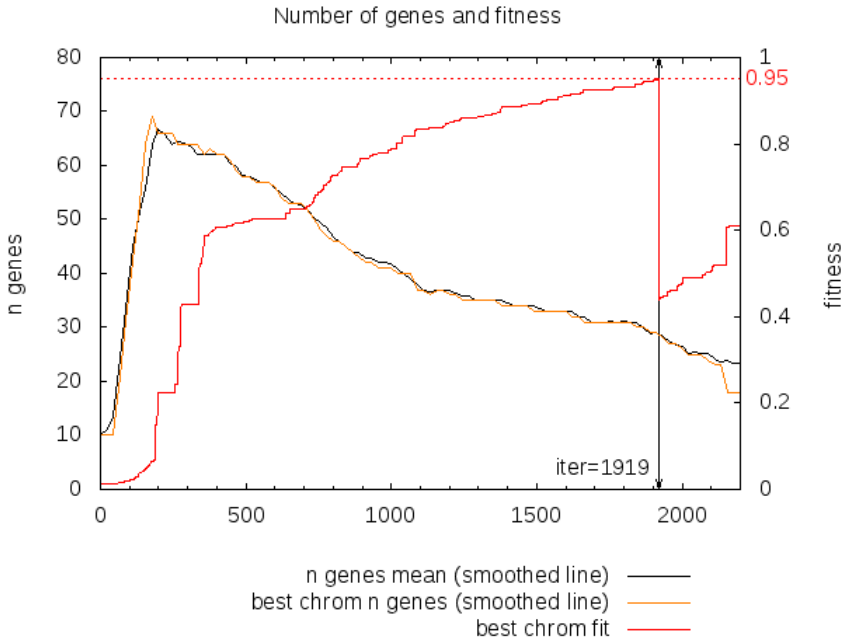


Figure 4.45: Number of genes and best chromosome fitness in phase 0.

In an initial stage of this phase, up to about iteration 180, the number of genes increases rapidly, up to a maximum that is ten times greater than the number of genes of the best chromosome generated at the end of this test (the final circuit synthesized in this test uses 7 genes, *i.e.*, 7 components). This fast increase in the number of genes is accompanied by a steep increment in fitness, which keeps increasing even after the number of genes starts to decrease (after about iteration 180). As mentioned before, this behavior seems to be a pattern, in this type of tests, for phase 0.

After iteration 180, the number of genes steadily decreases, and the fitness steadily increases until the threshold of 0.95 is reached. Note that at the end of this phase, the number of genes in the best chromosome is 29, which is (still) much higher than its final value.

Genes and fitness across phase 1

Phase 1 was dedicated to the reduction of the number of components in the circuit, and indeed this number was reduced from 29 to 14, as shown in Figure 4.46. Although this is a significant reduction, it still leaves the circuit with twice the number of components of its final version.

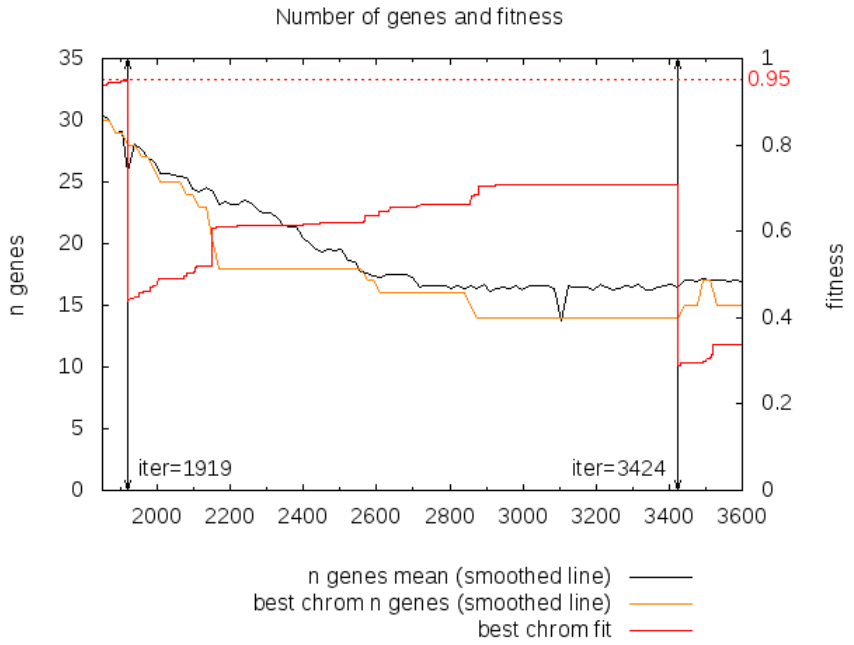


Figure 4.46: Number of genes and best chromosome fitness in phase 1.

In the transition between phase 0 and this phase, the fitness of the best chromosome has a sudden decrease, as expected, because of the changes in the fitness weights. During this phase, the fitness of the best chromosome increased consistently until stagnation occurred, which dictated the end of this phase at iteration 3424.

Genes and fitness across phase 2 and phase 3

Phase 2 was a long phase that lasted until iteration 104 879 and was dedicated to the reduction of the total MOS area used by transistors in the circuit. As mentioned before, during this phase the GA not only managed to reduce this area, but it also managed to reduce the number of components, which is shown in Figure 4.47. This reduction occurred at the beginning of this phase and brought the number of genes down to its final value of 7. It is also at the beginning of this phase that there is a significant increase in the fitness of the best chromosome, probably mainly due to this reduction in the number of genes.

For the remainder of this phase, the GA put a lot of effort into local exploration, trying not only to further reduce the total MOS area used by transistors, but also trying to improve other characteristics of the amplifier represented in the global fitness function. However, this effort is not evident in Figure 4.47, as there is little or no change in the traces just after

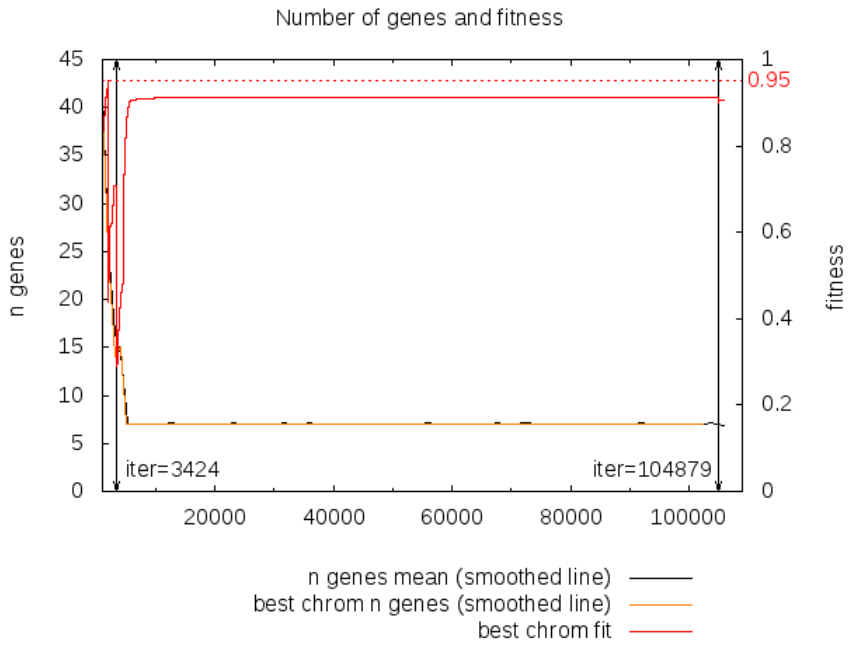


Figure 4.47: Number of genes and best chromosome fitness in phase 2.

iteration 3424. This lack of evidence of activity also occurs in the graphs of the number of genes and of the fitness of the best chromosome of phase 3, as shown in Figure 4.48.

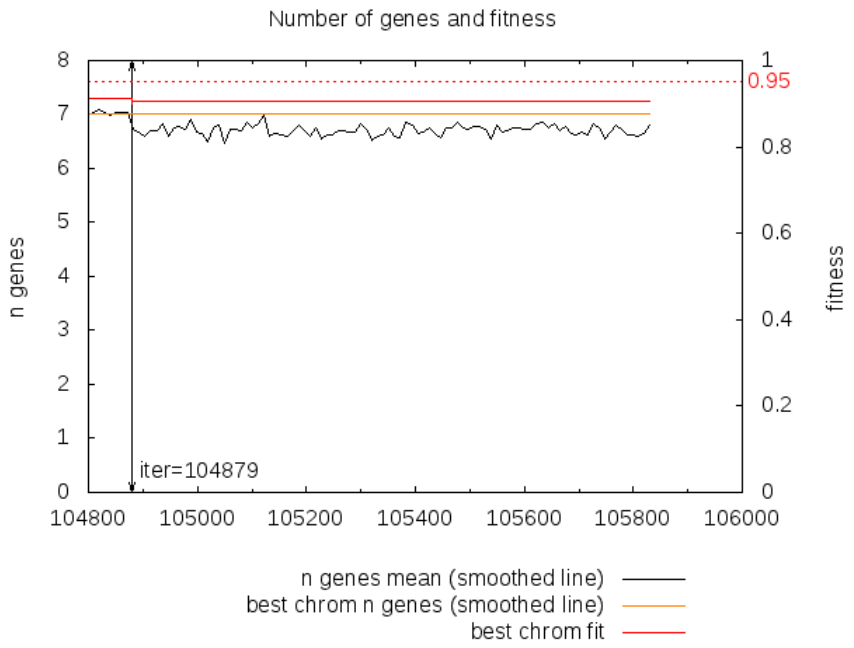


Figure 4.48: Number of genes and best chromosome fitness in phase 3.

Phase 3 lasted until iteration 105 831 and was dedicated to fine-tuning the frequency response of the amplifier. However, as mentioned before, few changes were made to the circuit since its version generated in phase 2, and the number of genes of the best

chromosome was kept unchanged during this phase, as can be seen in Figure 4.48.

Duration of a test

To get an idea of the order of magnitude of the duration of a test like this, it should be referred that this test lasted approximately 18 days. The run that generated the circuit presented in phase 3 took 237.5571 ks, which is about 2.75 days, and was performed using 16 cores of an "AMD EPYC 7501" type CPU (as were all the other runs of this test). It was a run selected from a set of very few successful runs produced by this test, where about several tens of runs (with different seeds used to initiate the pseudorandom generator) were attempted. Many of these runs were aborted before generating a useful circuit, either because the GA stagnated early, or because the intermediate circuit simulations began to take too long, preventing the GA from evolving in a timely manner. Another reason for interrupting a run is to exceed the time limit for using a specific computational resource. Many other runs simply do not converge on a useful circuit and stagnate somewhere before a useful solution is found.

Understandably, the duration of a test depends strongly on the machines that are available to run that test at a given time: if more machines (or more powerful machines) are available for a particular test, then it will certainly take less time to be concluded. But the point here is not exactly machine dependency: the main point here is that this type of testing takes a long time to be carried out within the reasonable and expected computational resources available for a research work like this one.

This is a severe limitation on the ability to perform exhaustive tests, such as those that would be necessary to try to optimize various parameters used by the GA. For example, both the optimization of the weights of the fitness function used on the set of phases of a run and the improvement of the heuristics used in the stagnation decision would be of great interest to try to increase the global evolution speed of the GA.

4.3.3.4 Using Fourier analysis to evaluate the amplifier output signal

In the aforementioned amplifier tests, the GA's ability to generate an amplifier with the desired gain was evaluated primarily through the measurement of the similarity between the signal obtained at the amplifier's output and the desired (or target) signal, both obtained through a transient analysis produced by NGSPICE (Section 4.1.1.2, eq. 4.3).

This process of evaluating the "quality" of the signal produced by the amplifier has the advantage of being relatively simple and fast to calculate. However, it does have some drawbacks. A non-zero $e_{V_{o_{rms}}}$ error is not easily related to measures of signal imperfection commonly used to rate the quality of an amplifier, such as Total Harmonic Distortion (THD). Moreover, while it is straightforward to employ the $e_{V_{o_{rms}}}$ error for scenarios in which the GA is tasked with generating an amplifier with a specific gain, it is no longer a simple matter to utilize it for problems in which the GA is required to produce an amplifier with a gain above a certain threshold, namely in a context in which a higher gain is better.

Another way to evaluate the "quality" of the signal produced by the amplifier is to perform a Fourier analysis after the transient analysis, both of which are performed by NGSPICE. Using data generated by a single Fourier analysis, it is possible to measure (at a given frequency) the gain of the amplifier, the offset of the output signal, and its THD. This method was compared to the previous one in terms of speed, and it was determined that the additional computational load was minimal, being offset by the duration of the remainder of the simulation. Consequently, this method was employed for a variety of tests throughout the course of this work, particularly on several of those described later in this document.

4.4 Amplifiers using MOS transistors with $L = 0.5\,\mu m$

This section presents additional examples of amplifier synthesis using the previously described techniques, complemented by a procedure for guided successive approximations. This procedure uses the circuit generated in one run of the GA as the embryo circuit for the next run (or set of runs). It changes the circuit's specifications in each run until a circuit with the desired specifications is eventually obtained. The first run uses a similar embryo circuit to those presented before. Subsequent runs use the circuit produced by the previous run. This procedure aims to obtain circuits with tighter specifications that could not be obtained in a single run. From experience gathered along this work with circuit synthesis using GAs, it seems that very demanding initial specifications make it very difficult for the GA to generate satisfactory results. Therefore, a progressive approximation with successive increments of complexity was tried.

In this section, the objective was to obtain an amplifier made with smaller L transistors ($L = 0.5\,\mu m$ was used), simulated with a more sophisticated SPICE model (higher than Level 1, preferably any BSIM3.3 variant), with a GBW of at least 30 MHz, preferably higher than 40 MHz, and using power supplies with lower voltages than those used so far (eventually as low as 1.6 V).

4.4.1 A 40 dB DC amplifier

In this test, an attempt was made to synthesize an amplifier using transistors with $L = 0.5\,\mu m$. These transistors were simulated using a Level 3 SPICE model with its default parameterization, except for the values of Kp and Vto, which were replaced by $Kp = 500\,\mu A/V^2$ and $Vto = 0.32\,V$ for NMOS transistors, and by $Kp = \frac{500}{3}\,\mu A/V^2$ and $Vto = -0.35\,V$ for PMOS transistors. These values produce better (more realistic) results than the model's default values, especially when considering transistors with dimensions of the order of those used in this test. However, they are not intended to be strict adaptations to any specific foundry technology.

The embryo circuit used in this test is very similar to the one used in the previous test, as shown in Figure 4.36. However, the value of the load resistors used in this test has been increased tenfold, hence $R_{Load1} = R_{Load2} = 200\,k\Omega$. By reducing the length of the transistors

to $L = 0.5\,\mu m$, it was expected that smaller transistors would be used in general, and a lower load driving capability was assumed.

As mentioned earlier, higher supply voltages seem to make it easier for the GA to synthesize amplifier circuits, so in this test V_{DD} and V_{SS} were kept at 5 V and -5 V, as shown in Table 4.15.

After 77 unsuccessful runs (each with a different seed), in the 78th run, at the end of phase 3, in generation 20385, the circuit of Figure 4.49 was produced. This circuit, although

Table 4.15: Circuit parameters (most relevant)

Description	Value(s)	Comment
Positive power supply	5.0 V	V_{DD}
Negative power supply	-5.0 V	V_{SS}
Voltage gain	100 V/V	100 V/V $\equiv$ 40 dB
Input signal for TRAN analysis	$f_0 = 2$ Hz $A = 10$ mV	$v_A(t) = A\sin(2\pi f_0 t)$
Input signal for AC analysis	$A = 1.0$ V $f \in [1\,\text{Hz} \,..\, 1\,\text{MHz}]$	$v_A(t) = A\sin(2\pi f t)$
THD @f_0	*NA*	Minimize
Bandwidth	500 kHz	
Max. input DC current	100 nA	Upper current limit for v_A input voltage source.
Max. power supply current	4 mA	Upper current limit for both V_{DD} and V_{SS} voltage sources.
Number of components	*NA*	Minimize
Model for MOS transistors	Level 3	Default parameters except for V_{t_0} and K_p
Transistor L	0.5 µm	Same L preset for all transistors
Area used by transistors	*NA*	Minimize
Transistors W/L ratio	W/L	1 to 500
Load resistors in embryo circuit	200 kΩ	$R_{Load1} = R_{Load2}$
Resistors in evolvable circ	$R_{g_i} \in [10\,\Omega \,..\, 10\,\text{M}\Omega]$	
Capacitors in evolvable circ	$C_{g_i} \in [0.1\,\text{pF} \,..\, 100\,\text{nF}]$	
Max. n of nodes in evolvable circuit	14	Excluding ground

it still has a rather useless component network consisting of C_1, C_2, R_{g_2}, $R_{aux2gnd_1}$, is an amplifier which, when evaluated (by NGSPICE) with a load capacitor of 10 pF, has a gain of

35 dB, a GBW product of 22 MHz and a Figure of Merit (FoM) of about 130 MHz pF/mW, where FoM is given by 4.9, in which GBW is the Gain-Bandwidth product of the amplifier (in MHz), C_{Load} is the test load capacitance (in pF), and Pwr is the power supplied to the circuit (in mW).

$$FoM = \frac{GBW \times C_{Load}}{Pwr} \tag{4.9}$$

It should be noted that in this test, unlike in previous tests, the GA did not use the auxiliary polarization circuit consisting of the $M_{p_{bias}}$ and $M_{n_{bias}}$ transistors of the embryo circuit.

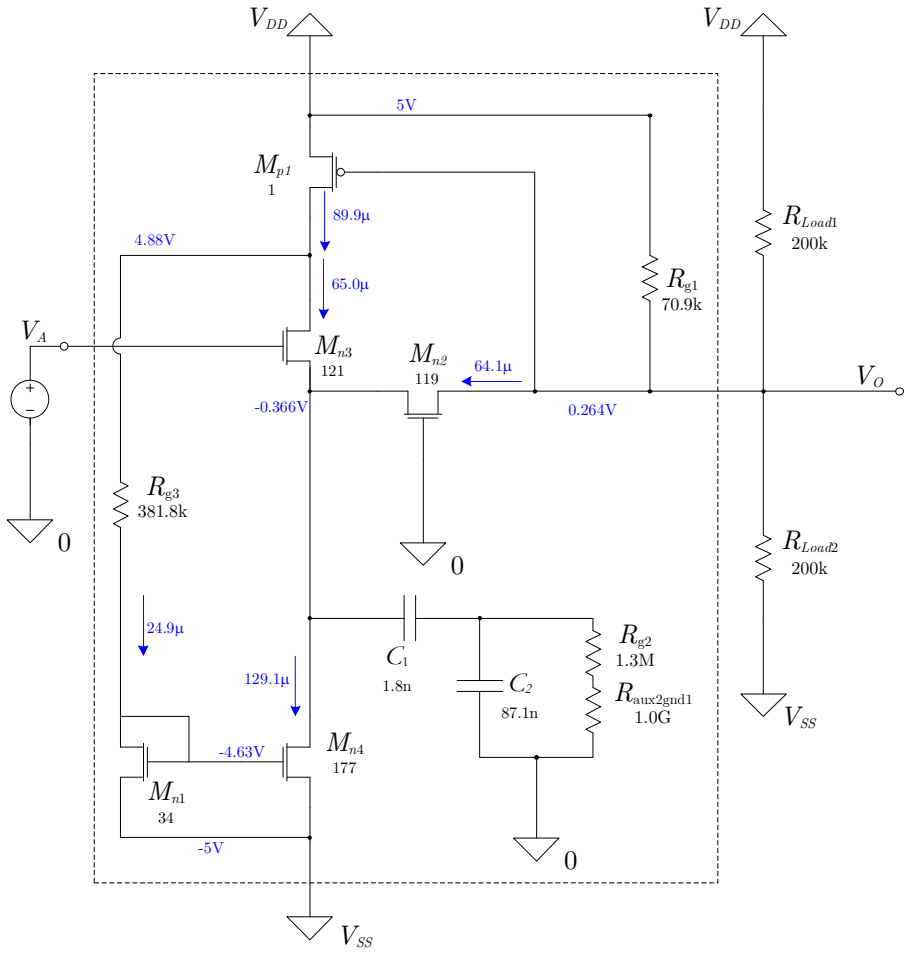


Figure 4.49: Circuit generated at the end of phase 3.

Tuning some specifications

Now that the GA has managed to synthesize an amplifier that fulfills some of the objectives, the circuit's supply voltage is gradually lowered until the intended value of $V_{CC} = -V_{SS} = 0.8\,V$ is (eventually) reached. This is done by resetting the GA to its initial state, setting the new embryo circuit to the last synthesized circuit, and lowering the power supply by a small Δ amount. Then let the GA evolve and, eventually, produce a better circuit. If the circuit is better than the previous one, then repeat the process until it is needed. If not, go back, and reduce the Δ and iterate. This process is repeated (adjusting the Δ as appropriate

and, eventually, changing any other characteristic of the circuit if required) until either the desired power supply voltage is achieved or too many iterations are elapsed, in which case the last circuit achieved is considered the best. This successive approximation procedure can be carried out with varying degrees of automation, depending on numerous factors. In this work, the progressive decrease in supply voltage was initially tested manually and then automated and integrated into the GA kernel. Applying this iterative process of successive approximations to the circuit of Figure 4.49, the circuit of Figure 4.50 was produced.

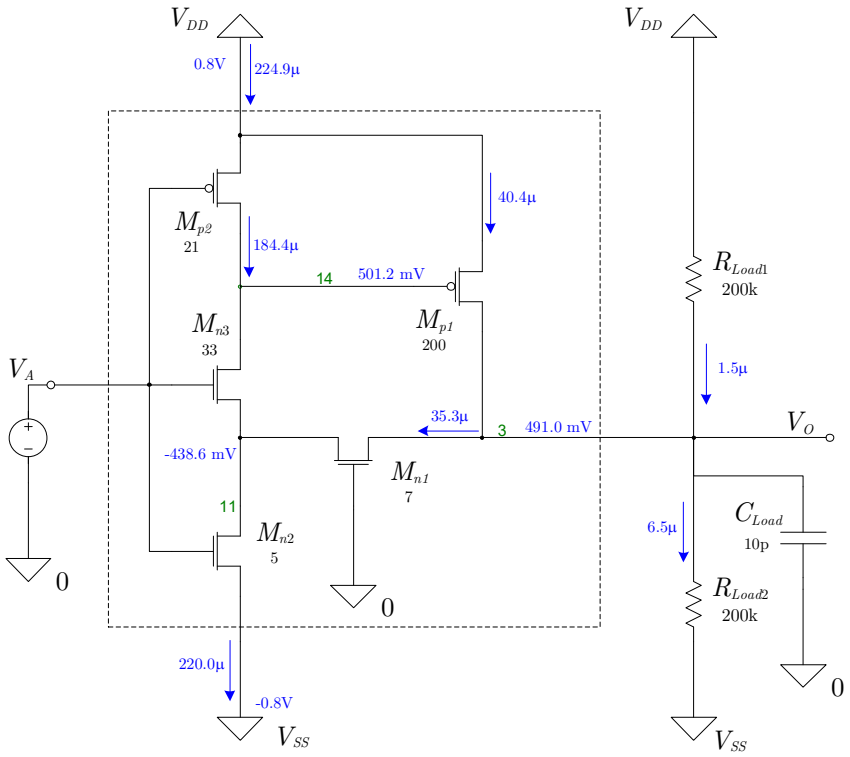


Figure 4.50: A 40 dB amplifier using transistors with $L = 0.5\,\mu m$.

The circuit produced by the GA shown in Figure 4.50 is derived from the one shown in Figure 4.49 and was obtained after a few steps (of the successive approximations procedure) in which not only the supply voltage was progressively reduced, but also a BSIM3.3 model was introduced to model the transistors (with a parameterization close to a standard 130nm technology). This circuit is also an amplifier (as its predecessor of Figure 4.49), which, when evaluated by SPECTRE/CADENCE 6.0, using a standard 130 nm technology, has the results depicted in Table 4.16.

This circuit meets the objectives. Therefore, from the point of view of circuit synthesis with GAs, it is a success case. The proof of concept has been achieved: if auxiliary techniques such as those proposed here are used, GA synthesis of analog amplifiers is feasible without using any previously known building blocks. But this synthesis task was found to be computationally heavy; therefore, other features often found in sizing EDA tools were not incorporated, such as Monte Carlo analysis and layout-aware simulation [Lourenço et al., 2016]. Monte Carlo analysis is commonly accepted as a standard method to estimate

circuit yield, but it is also known to be highly time-consuming. Although some techniques to reduce the computational burden have been described in the literature [Canelas et al., 2020], their inherent addition to the overall computation times was still considered not tractable. For similar reasons, a layout-aware simulation was not incorporated in this work (although related accelerating techniques may eventually be useful [Domingues et al., 2022, Hakhamaneshi et al., 2019]).

Table 4.16: Results obtained for the 40 dB amplifier of Figure 4.50.

Description	Value(s)	Comment
Positive power supply	0.8 V	V_{DD}
Negative power supply	−0.8 V	V_{SS}
Voltage gain	102.3 V/V	102.3 V/V ≅ 40.2 dB
THD@1kHz	0.67%	$V_{i_{ampl}} = 1\,\text{mV}$ $V_{i_{os}} = -6\,\text{mV}$ $V_{o_{ampl}} \approx 102.5\,\text{mV}$
Bandwidth	684 kHz	$f_L = 0$ $f_H \approx 684\,\text{kHz@}-3\,\text{dB}$
GBW	70 MHz	
Input DC current	≈ 0	
Power supply current	2 × 243.6 µA	$I_{DD} \approx I_{SS} \approx 243.6\,\mu\text{A}$
FoM	1436 MHz.pF/mW	$C_{Load} = 10\,\text{pF}$
Number of components	5	
Total transistor area	66.5 μm^2	Estimated area used by transistors.

CHAPTER 5

Conclusions and Future Work

This chapter presents the conclusions and suggestions for future research effort that result from this work.

The main research topic of this work is to better understand whether it would not be possible to go further with GAs in the automatic synthesis of circuits without using any previously known building blocks, as a way of promoting novelty in topology generation. For some years now, the synthesis of digital circuits has been almost fully automated, but in analog circuit synthesis the topology generation stage still requires a lot of human intervention. A significant part of the research effort devoted to topology synthesis uses EAs, and a smaller part uses GAs, usually trying to simultaneously generate an appropriate topology while sizing the circuit components. However, the use of GAs in circuit synthesis has gradually stagnated since the beginning of this millennium.

In the area of (automatic) analog circuit synthesis, many research works, either using EAs or other paradigms, use some kind of knowledge-based synthesis, typically in the form of building blocks. However, this knowledge-based synthesis tends to limit the novelty of the generated topologies. Additionally, a significant number of works dedicate effort to generating passive circuits (the synthesis of filters is very common in the literature), but the generation of active circuits, such as amplifiers, is much less often attempted.

It was not clear (to the author) why better results have not been obtained with GAs in the synthesis of analog active circuits without the use of building blocks, namely for amplifiers,

so this work followed the work of many others trying to contribute to answer this question, which is closely intertwined with the main research question of this work (and its derived questions) presented in Section 1.2.

5.1 Answering the main research question and its derived questions

The main research question adopted in this work was:

> Is it possible to generate new circuit topologies for CMOS amplifiers, at flat circuit-level, in an automatic way (software generated), with optimized specifications and high efficiency at submicron technologies?

And the proposed hypothesis to address this problem and to guide the research effort was:

> It is possible to develop a methodology and a software tool based in GAs to, automatically, generate novel flat circuit-level topologies for CMOS amplifiers with high efficiency at submicron technologies.

It can be considered that the research question has been partially answered over the course of this work, in the sense that the results obtained, particularly for amplifiers, make it reasonable to say that it is indeed possible to automatically generate new circuit topologies, namely for CMOS amplifiers, at flat circuit-level, using GAs. However, regarding the level of optimization over the specifications, a more cautious answer is in order, since there was no exhaustive testing about the simultaneous optimization of several characteristics of the amplifiers. In fact, it can be considered that the only specification that was optimized was the area used by the transistors in the circuit (which the GA tried to minimize). Although it is possible to foresee how other specifications might be considered in the GA, it is not easy to predict how they would affect its performance, let alone the quality (usefulness) of the generated circuits. And in terms of the level of efficiency achieved, once again, a cautious response is required, since it depends on what is meant by "efficiency". If it is understood to mean making the most of the area used by components, particularly transistors, then it can be assumed that it should be possible to achieve good efficiencies, depending on the weight given to them in the fitness assessment. However, if it is a question of making the most of other characteristics, with the aim of optimizing all of them at the same time, then there is again the question of whether this is actually possible and how the overall performance of the GA will be affected.

The research question (and the proposed hypothesis) raised several other questions, most of which were addressed in the course of this work. These derived questions, already presented in Section 1.2, are reviewed in the following paragraphs.

Three of these derived questions have answers that are closely related, so they will be discussed together.

- To what extent does the GA-c is suited to circuit synthesis?
- What circuit codification is best suited for analog circuit synthesis, namely for amplifier topology synthesis?
- How can VLCs contribute to the GA's ability to encode circuits, and to what extent does it affect the quality of the topologies produced?

As part of this study there was the intention to understand the main constraints of GA-c for this task and to investigate possible enhancements that make these algorithms more suitable for circuit synthesis.

Using GA-c and a simple encoding scheme for circuit representation based on bit strings, the algorithm was indeed able to generate several simple digital circuits, such as 2-bit NAND and NOR gates. In these tests, only transistors were available for circuit synthesis, and they were all pre-sized (their W and L dimensions were constant throughout the evolution). This eliminates one task of the synthesis, the sizing task, leaving the GA "concerned" only with the topology generation task.

Even so, scalability for more complex circuits proved to be an impractical task, and despite the research effort expended, which amounted to several thousand hours of CPU time and the equivalent in human effort, the GA was never able to generate circuits of higher complexity, such as an XOR gate or a 2-bit half-adder.

The research effort then focused on the use of VLCs, which required the use of a new circuit coding technique. This shift in the research effort implied several changes and enhancements to the original algorithm, namely in the crossing and mutation operations.

Using VLCs, the GA was able to synthesize circuits that it had not been able to synthesize before. In the digital circuits field, it was able to synthesize the 2-bit XOR gate and the 2-bit half-adder. And in the analog field, in which all attempts to synthesize analog amplifiers had been a complete failure, since even the synthesis of amplifiers with gain 6 dB had been unsuccessful, the GA was finally able to synthesize circuits implementing DC amplifiers with gains up to 40 dB. In fact, the best circuit synthesized by the GA is an amplifier with a voltage gain of 102.3 V/V $\cong$ 40.2 dB, a GBW of 70 MHz, with a FoM of 1436 MHz.pF/mW (measured in SPECTRE/CADENCE 6.0).

From the knowledge gained along the course of this work about the behavior and performance of the GA when using FLCs and VLCs, it seems reasonable to say that VLCs are much better suited for circuit synthesis than FLCs, particularly for circuits beyond a certain complexity, which automatically excludes GA-c as a good option for this task.

Another finding that distinguishes the behavior of the GA when using VLCs is that it shows low sensitivity to the number of components in the embryo circuit, which is quite different from when using FLCs. This is an important robustness factor for the GA.

Other questions derived from the main research question (also presented in Section 1.2)

were the following:

- How multi-objective algorithms can contribute to better accomplishment of amplifier specifications?
- How parallelization, and what kind of parallelization, can contribute to the outcome of a GA when applied to circuit synthesis.

The use of Multi-Objective Evolutionary Algorithms (MOEAs), such as NSGA-II proposed in [Deb et al., 2002], was not addressed in this work, so the first question cannot really be answered based on the results of this work. Typically, in multi-objective optimization problems, there are multiple conflicting objectives that have a set of Pareto optimal solutions. In these problems, improving one objective can lead to the degradation of another, so there is no single solution that can optimize all objectives simultaneously. Instead, the best trade-off solutions, called Pareto optimal solutions, are selected by a decision maker (often a human or some kind of human-guided process). The use of such algorithms is probably the natural progression for future work in this research.

Unlike the MOEAs question, the parallelization question was addressed in this work, since parallelization was actually used and even more than one model was tested. GAs are highly parallelizable, and it is essential to take full advantage of this feature in order to reduce the overall execution time, which is very high when GAs are used for circuit synthesis. The fitness evaluation stage of each circuit is particularly computationally intensive, and it is at this stage that parallelization should be focused.

Currently, there are several parallelization models whose implementation depends not only on the computing platforms but also on the problem to be solved. In this work, the parallelization was done at the thread level, i.e., the implementation of the genetic algorithm was done through multi-threaded programming, using the thread pool variant of the boss-worker model. The desired effect was to run the simulation (in NGSPICE) of a circuit on every core available on the machine and to keep all cores busy until all chromosomes in the population have not been evaluated.

Of course, other parallelization models can be used, properly adapted to the computational platform under consideration, as long as the focus is on the fitness evaluation stage.

Embedded in the challenge of circuit synthesis with a GA is, inevitably, the pragmatic and ever-present problem of the computational power effectively available for a given research work, which inexorably limits the results obtained through GAs. And certainly a major drawback faced during this work was the lack of more powerful computational resources. But this is ultimately a neutral (perhaps even an empty) argument since it must be an omnipresent argument in many research works. Nevertheless, the idea persists that more and better answers would have been possible if more computing resources had been available. And ideally, with significantly more computing resources available, it might have been possible to answer the question of what really limits GAs in circuit synthesis:

whether it is a limitation inherent in the algorithm itself, or whether it is simply insufficient computing power.

5.2 Enhancement techniques

In the course of this work, some techniques for improving the GA were developed, which were important for obtaining the results presented in this thesis. Some of these techniques may be more relevant than others, and some are surely easier to implement than others, but all have certainly contributed to obtaining these results. These techniques are briefly summarized below.

Avoiding floating nodes

It is very common to have components with floating nodes in circuits generated by the GA. These circuits are usually rejected by SPICE-like simulators because the analysis methods used by SPICE require that every node in a circuit has a DC path to ground.

One technique commonly used to deal with this problem is to connect each of these nodes to a predefined node (often to ground) with a high-value auxiliary resistor (often with a 1 GΩ resistor). This technique has been implemented in this work, and these auxiliary resistors are in fact very often found in intermediate circuits during the evolution of the GA, although they rarely find their way into the final circuit.

Another related problem, often found in digital circuits generated by the GA, is the presence of MOS transistors whose gates are virtually floating in some circuit states. This happens, for example, when the gate of a particular transistor is connected only to the drains of other transistors, and there are circuit states in which these transistors are all "off", leaving that gate connected only to very high impedances, leaving it in an unstable logic state, highly susceptible to interference from unwanted effects such as noise and leakage currents.

In order to reduce the occurrence of this type of virtually floating node, particularly in the case of MOS transistor gates, the previous technique was adapted to these cases by connecting these nodes via high-value resistors to pseudo-noise voltage sources, which reveals the instability of the state of that transistor. This proved to be a very effective procedure.

Adaptive probability of chromosome acceptance

Yet another enhancement technique is a heuristic used to implement an adaptive acceptance probability of mutated chromosomes according to a measure of the quality of the solutions obtained along the evolution. This technique improves the local exploration capabilities of the GA and reduces its sensitivity to the parameterisation of the mutation probability.

Segmented evolution

In addition to the techniques already mentioned, a procedure for segmenting the evolution

of the GA was developed, using several phases with different parameterizations, which proved to be a very valuable technique for steering of the GA towards subgoals.

"Circuit cleaning" technique

Another enhancement is a "circuit cleaning" technique that eliminates redundant and useless components in the circuit. In this technique, the weight of a penalty function is dynamically adapted to steer the algorithm towards the sub-goal of minimizing the number of components. This weight is dynamically adapted so that in the early stages of the circuit evolution it is low enough to allow a great exploration capability of the GA, but high enough in later stages so that useless components are effectively removed from the circuit. The contribution to the results obtained in this work from the combined use of this technique and the previous one ("Segmented evolution") is perhaps one of the most important of all.

VLCs and enhancement techniques together

The use of VLCs in conjunction with all of the techniques mentioned above has allowed the synthesis of more complex digital and analog circuits, providing the desired scalability of the entire synthesis process achieved by the GA, which was lacking in the early stages of this work. This is an important result of this work, possibly the most important: VLCs, used in conjunction with the aforementioned techniques, were crucial to the GA's success in generating the most relevant (least basic) circuits presented in this work.

5.3 Other considerations

Many considerations naturally arise in a work of this kind. Some of them, as well as some suggestions, are presented in the following paragraphs.

Restrictions in simulation model level

In many runs of the GA where synthesis from scratch was intended, a SPICE *LEVEL 1 MOSFET* simulation model was used for MOS transistors because the time penalty for using a higher level, such as *BSIM3*, was unaffordable given the available computer power for this research study.

However, some of the final circuits synthesized in the context of this work were generated while using a *BSIM3* model in the simulations performed by the GA (namely the amplifier shown in Figure 4.50), but in reality this model was not used from the beginning of the evolution: at the beginning of the evolution a level 1 model was used and then in the final part a *BSIM3* model was used, but only after a "partially working" circuit had already been obtained with the level 1 model. This process allowed the generation of circuits that work correctly in simulation when tested with standard models of 130nm technologies (using SPECTRE/CADENCE 6.0); these circuits were generated by the GA with much less CPU time (compared to generating from scratch using *BSIM3*).

An alternative to this process might be to use a faster model by default (probably the level 1 model) and occasionally, with some probability, use a higher level model, such as the *BSIM3* model, as a second test for a circuit, and if the (relevant) behavior is quite different between the two simulations (faster but lower level model versus slower but higher level model), then that chromosome (circuit) would be scored with low fitness. The balance between the probability of using the higher level model and the total execution time should then be adjusted to meet the constraints of the ongoing research study (taking into account time constraints versus available computing power).

Flat circuit-level only?

The main research question of this work embeds the restriction of not using simple topologies in the synthesis process, *i.e.*, unlike other works, it was not considered that well-established simple topologies could participate as building blocks (or as super-components, as they are sometimes called). For example, in addition to transistors and resistors, other topologies could be used as if they were a component, like current mirrors, differential pairs, and others. If on the one hand this feature may reduce the drive for new topologies generation, on the other hand it may increase the success rate of synthesis of circuits that fulfill the intended objective. In some contexts, having the option of balancing these options can be beneficial.

Number of nodes in the circuit

In all tests, the number of nodes allowed in the evolvable circuit was fixed and predetermined by the user. This was done heuristically, and it remains unclear how restrictive this setting is for the global behavior of the GA in the synthesis of a particular topology. This is a topic that requires further work to know how this affects the results, in particular how it affects the convergence speed of the GA. In future tests, an adaptive number of nodes should be tried *i.e.*, this number could start relatively low and increase only as needed.

Parameter dependency

Although some automatic adaptive parameter techniques have been developed in this work, there are still numerous parameters of the GA that require manual settings, and these settings can affect the overall performance and outcome of the algorithm. This is an important issue because excessive sensitivity to some parameters has a negative impact on the robustness of the GA. It is also true that knowledge about parameter tuning comes from experience, and this experience may take a long time (and a lot of CPU time) to reveal itself. Nevertheless, this is undoubtedly a topic that needs further work.

5.4 Final comments

Circuit synthesis by means of GAs, especially at flat circuit-level, is a computationally expensive task, arguably too expensive for today's standard computing power, but GAs are capable of synthesizing useful analog circuits, and when used in the absence of building

blocks they have an intrinsic capability of novelty in topology generation, which is a good motivation to devote research efforts to study their contributions to the ADA field. This work is a contribution to this field, in particular to the research efforts made in this area with GAs, and by no means intends to present a finished solution for the automatic generation of analog circuits. There is still a lot of work to be done, and it seems that it will be a long time before we have a tool that completely automates the automatic generation of analog circuits.

APPENDIX A

Appendix: Crossover and mutation probability tests

This appendix contains data in graphic form produced by tests performed with the intention of tunning some parameters of the GA, namely the probabilities of crossing and mutation. These tests use a third degree polynomial for which the genetic algorithm is requested to find the coefficients (as described in Section 3.2.2.1). To help understanding how those probabilities affect the convergence of the GA in this problem, these tests systematically scan some chosen ranges of those probabilities.

Two sets of tests were conducted: in the first set the mutation probability is constant during evolution of the GA and in the second set the mutation varies from an initial higher value to a smaller value in the end of the GA run. The graphs generated by the data produced by those tests are presented in the following sections.

Each graph contains the following traces:

- The best chromosome fitness.
- The fitness average in the population.
- The number of chromosomes that needed fitness evaluation.
- A moving average of the number of chromosomes that needed fitness evaluation.
- The average of the number of chromosomes that needed fitness evaluation calculated for the entire run.

These graphs are presented in groups of three because each pair of crossover and mutation probability values was run three times, each with a different random initialization of the first generation of the GA. The pairs of crossover and mutation probability values

cover all combinations of the sets of values scanned. These sets of values (eq. A.2 and eq. A.1) were chosen because they were considered "useful" or "interesting" values based on experimental knowledge attained so far.

$$p_{cross} \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\} \tag{A.1}$$

$$p_{mut} \in \{0.01, 0.02, 0.04, 0.08, 0.16, 0.32\} \tag{A.2}$$

A.1 Constant mutation probability test set

The set of tests in which the mutation probability is constant during evolution of the GA produced the graphs shown in sections A.1.1 to A.1.7, which results from sequentially scanning the values in eq. A.1. Each section contains the graphs for all mutation probabilities in eq. A.2.

A.1.1 Crossover probability 0.3

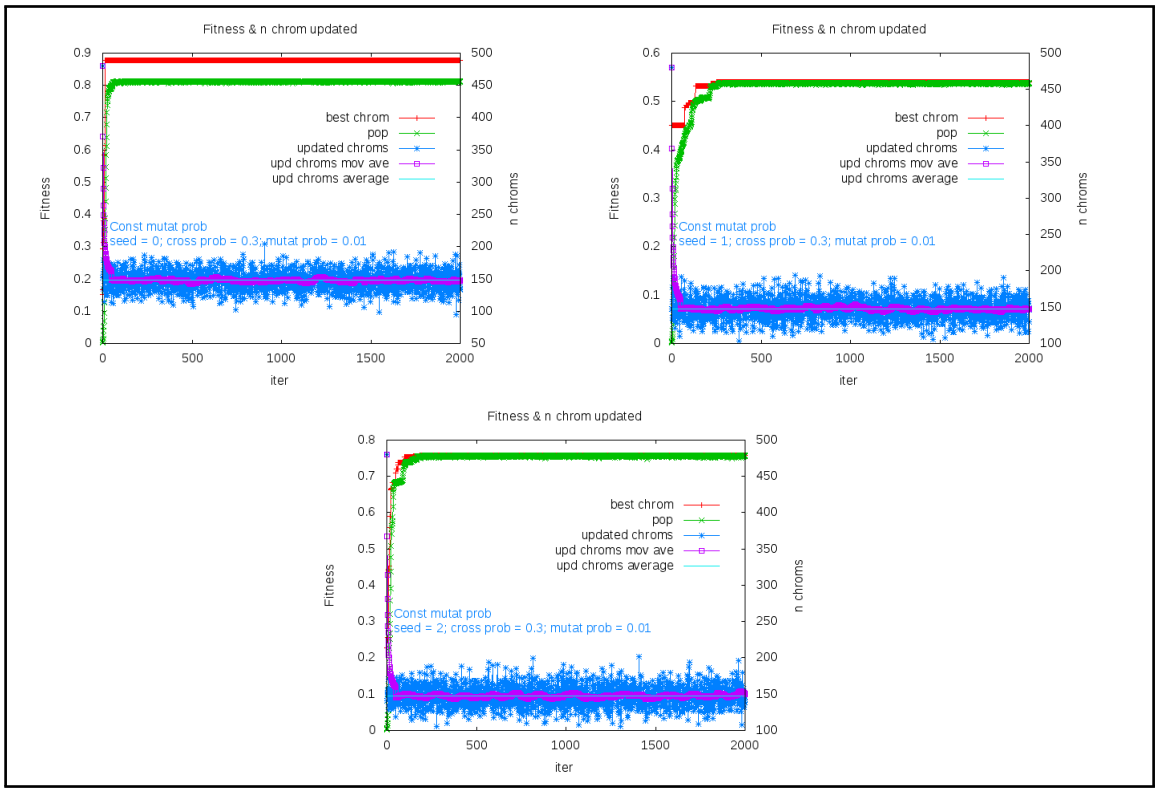


Figure A.1: Constant mutation probability 0.01

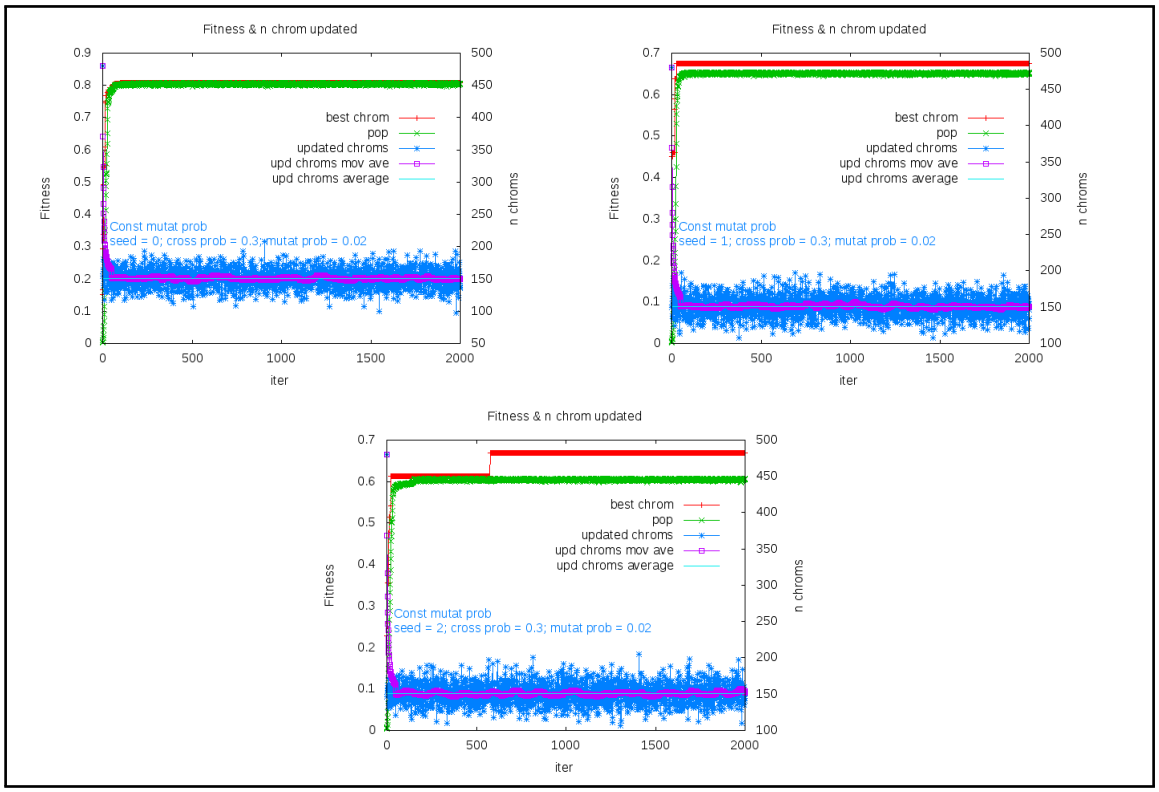


Figure A.2: Constant mutation probability 0.02

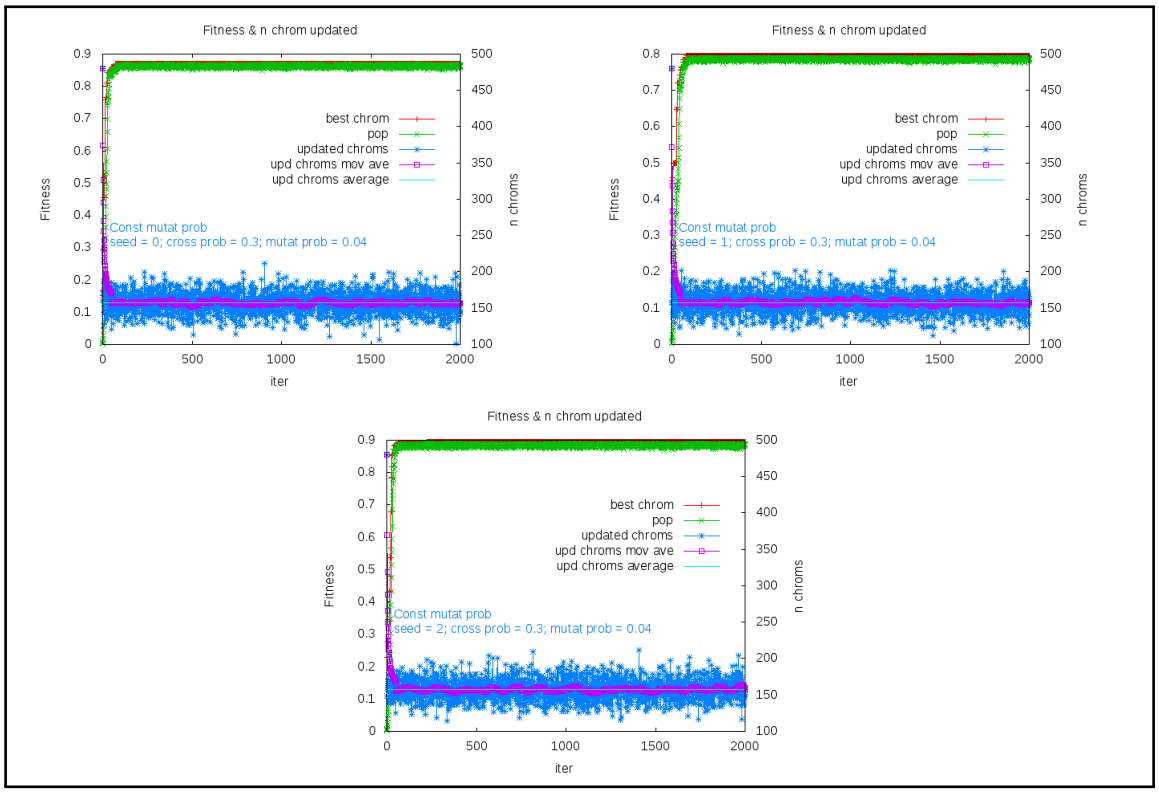


Figure A.3: Constant mutation probability 0.04

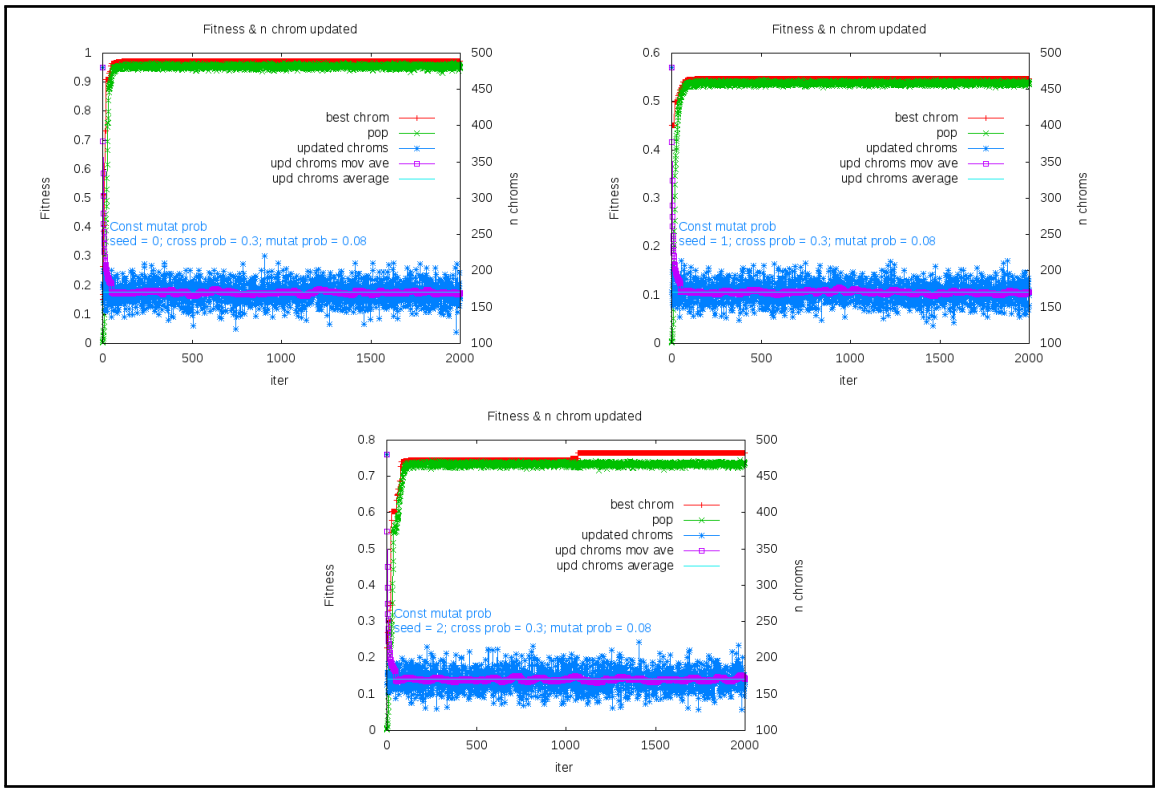


Figure A.4: Constant mutation probability 0.08

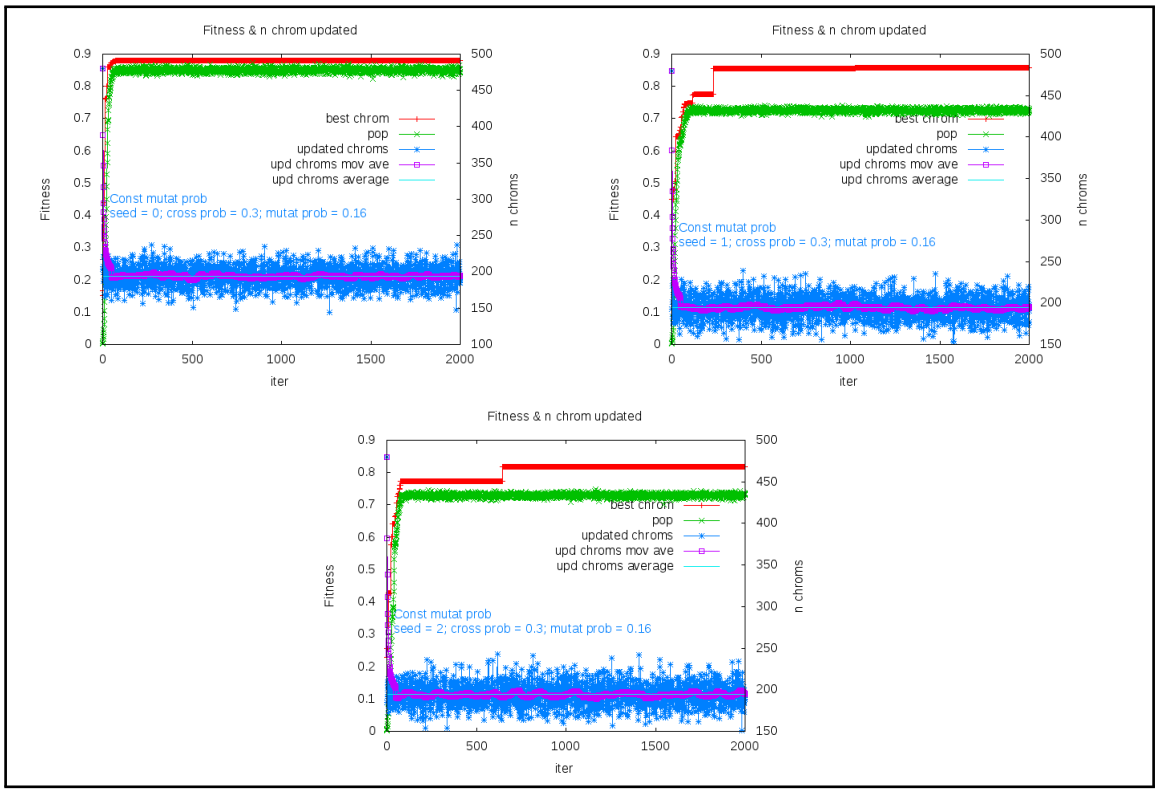


Figure A.5: Constant mutation probability 0.16

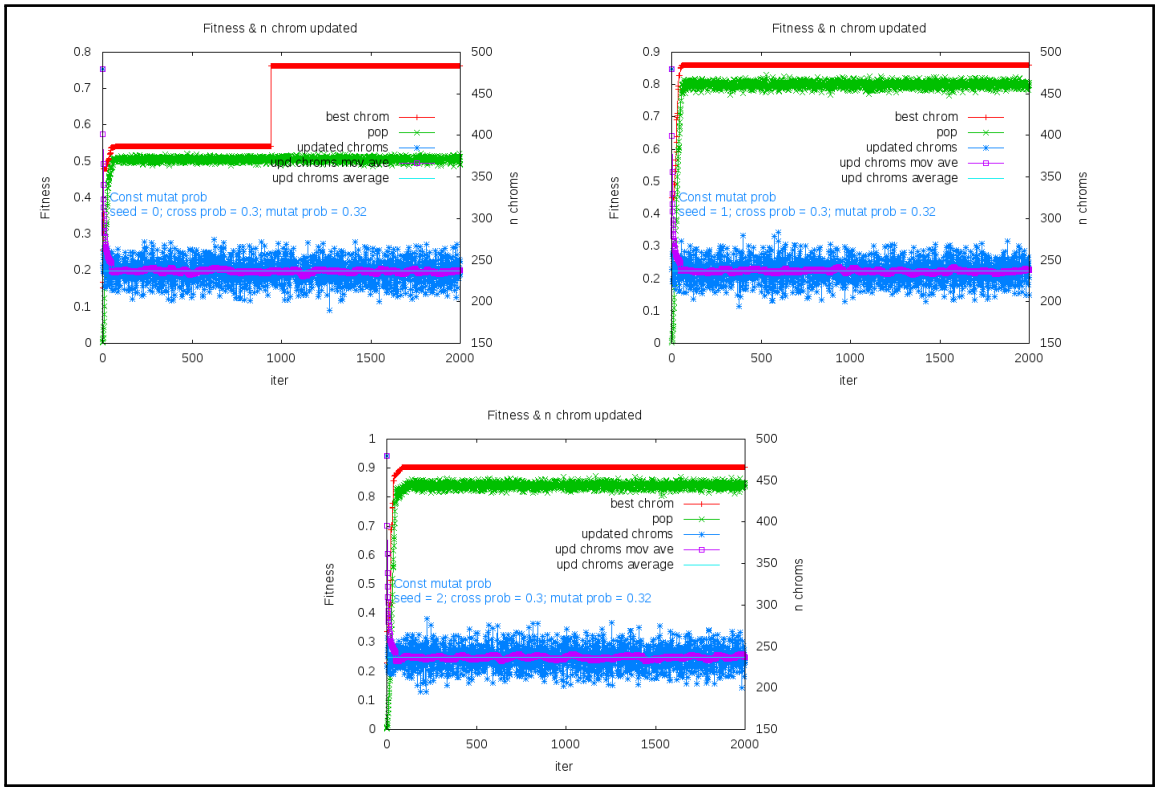


Figure A.6: Constant mutation probability 0.32

A.1.2 Crossover probability 0.4

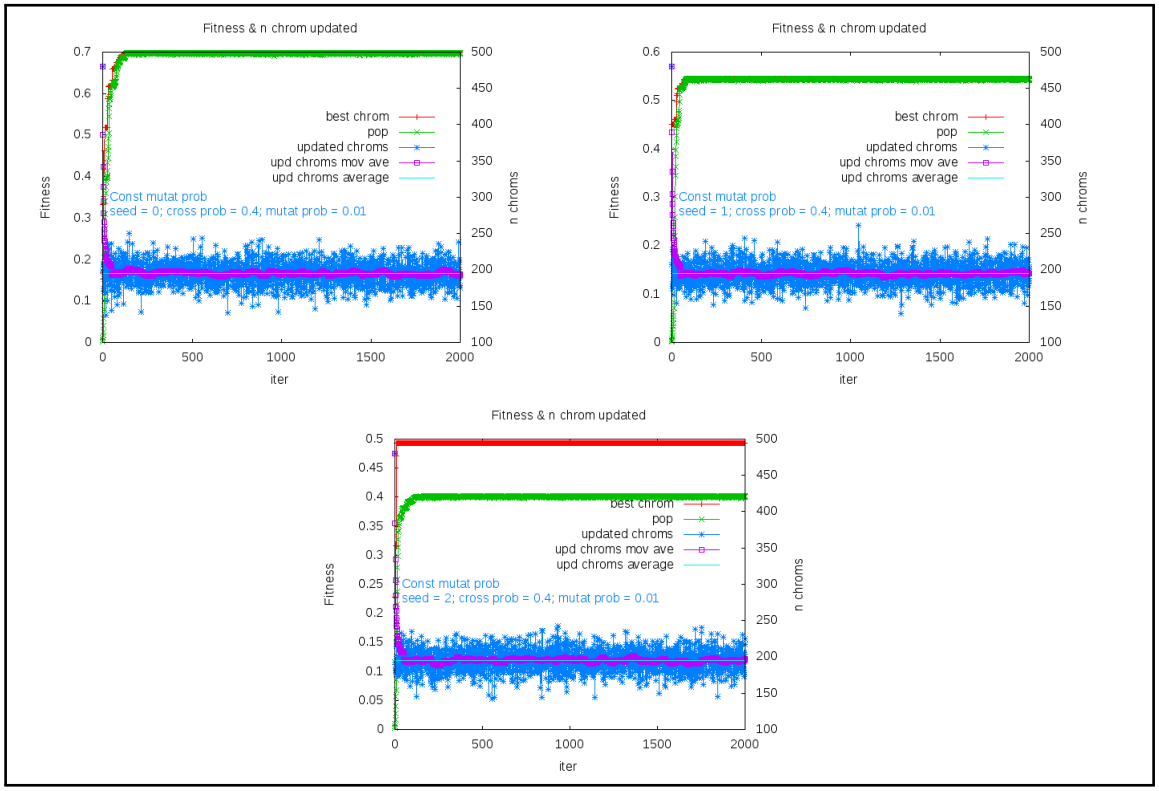


Figure A.7: Constant mutation probability 0.01

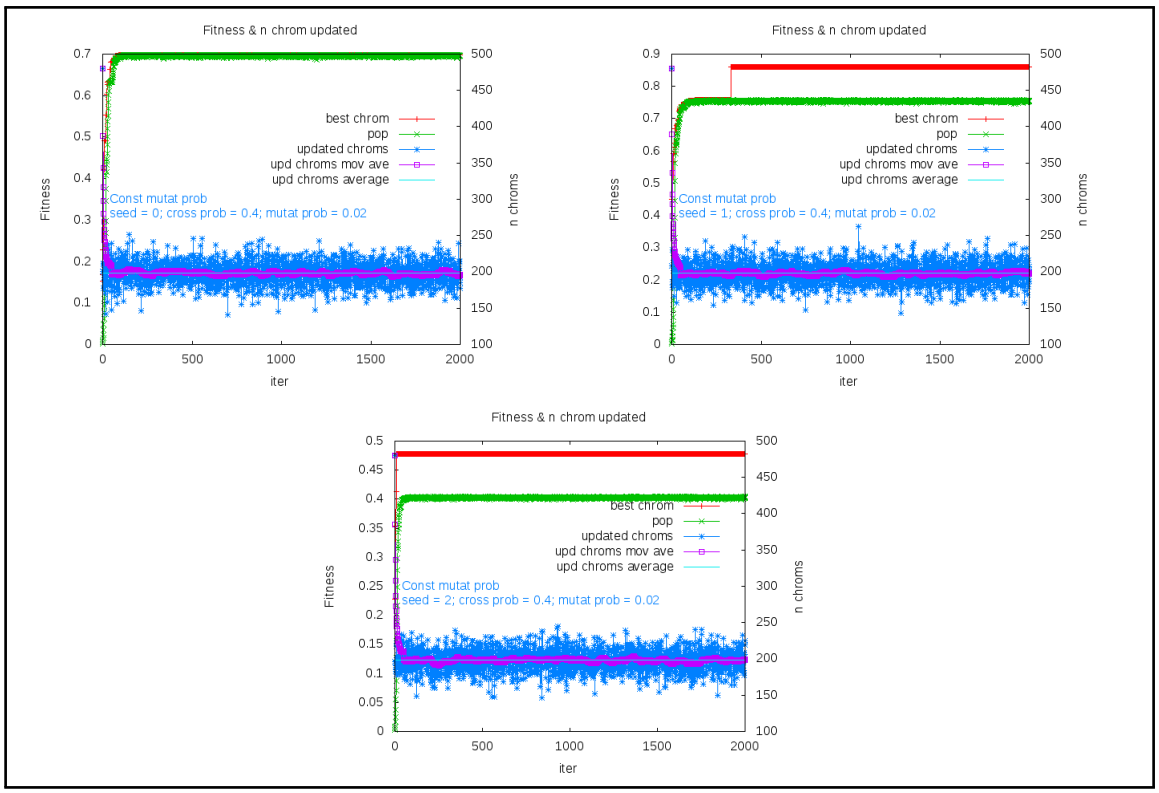


Figure A.8: Constant mutation probability 0.02

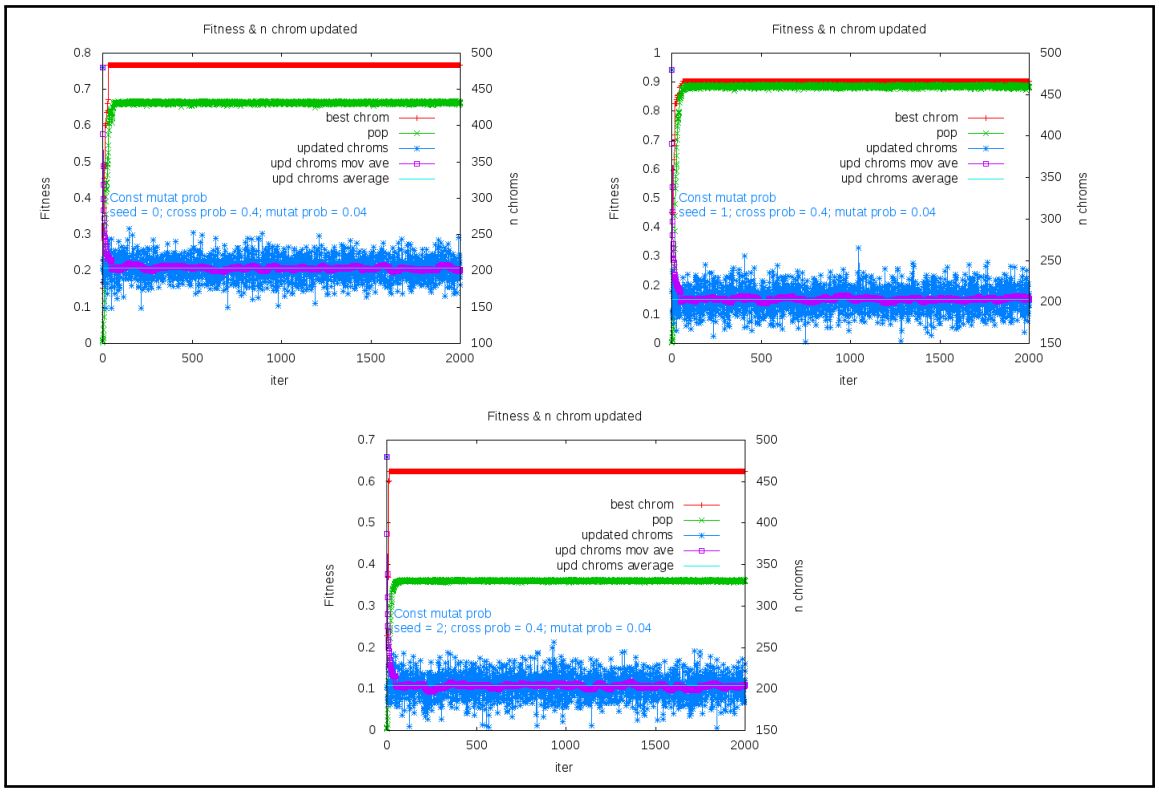


Figure A.9: Constant mutation probability 0.04

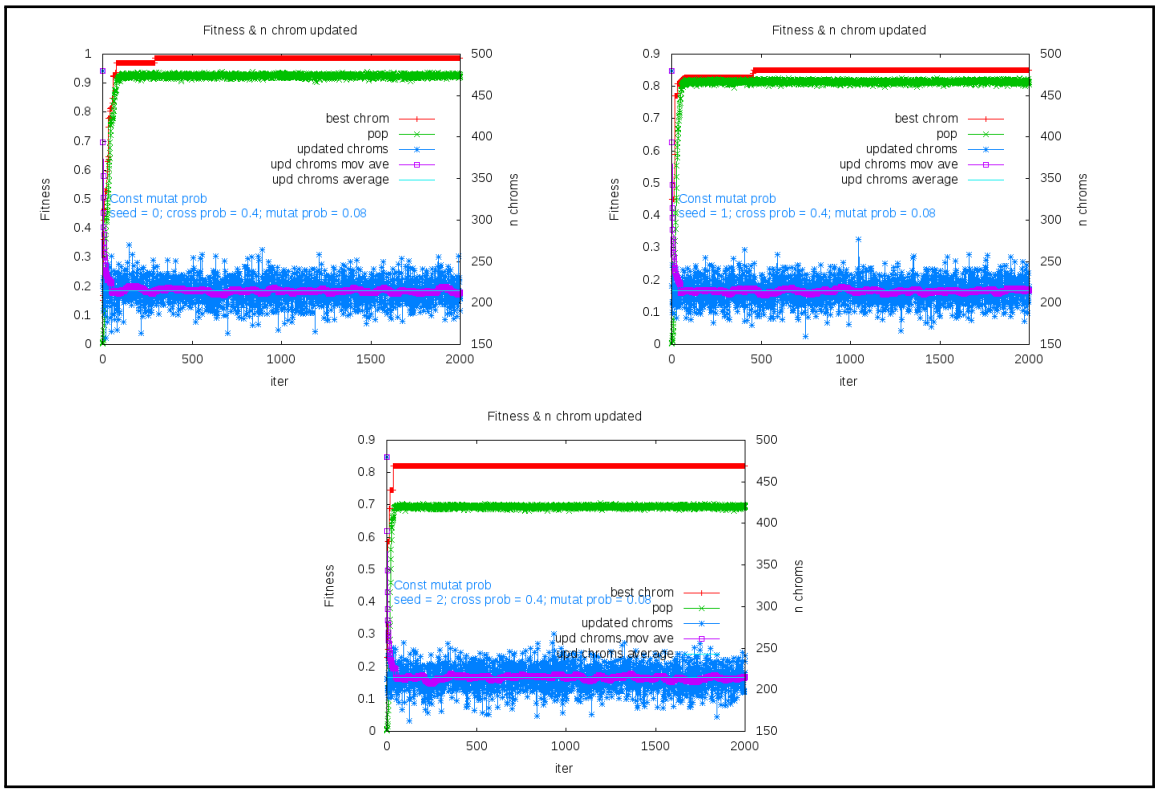


Figure A.10: Constant mutation probability 0.08

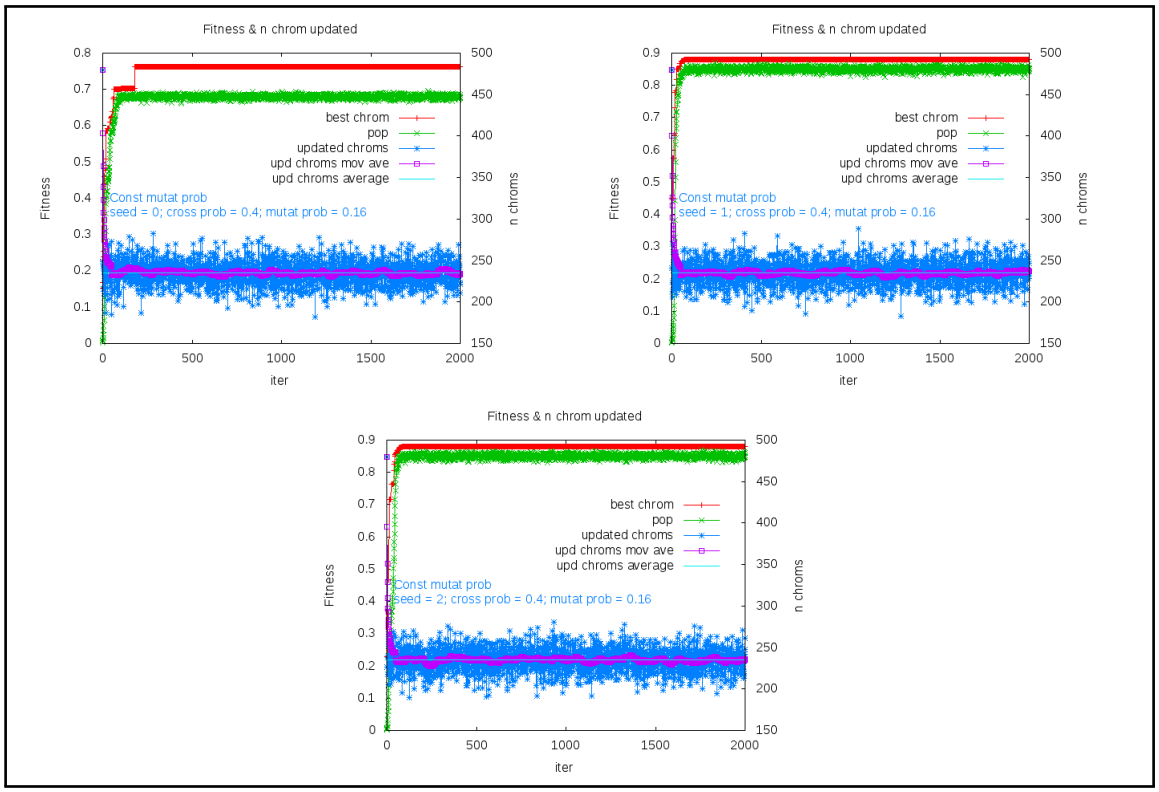


Figure A.11: Constant mutation probability 0.16

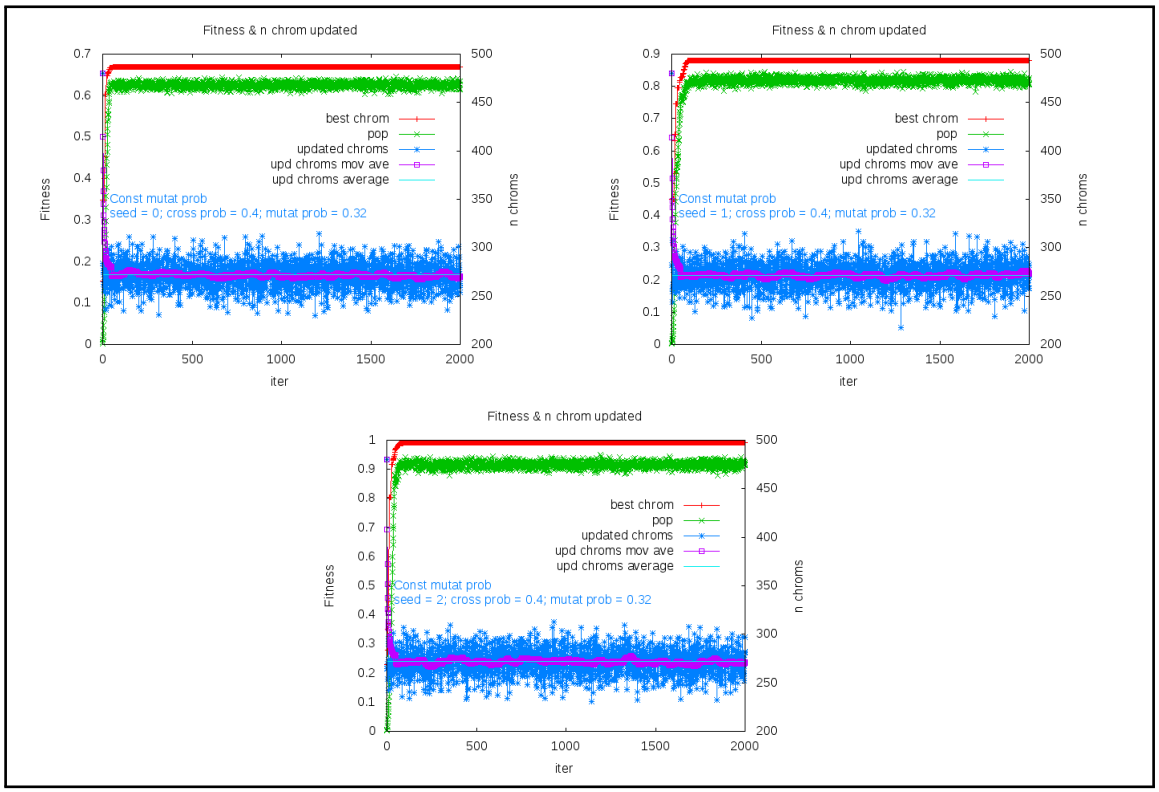


Figure A.12: Constant mutation probability 0.32

A.1.3 Crossover probability 0.5

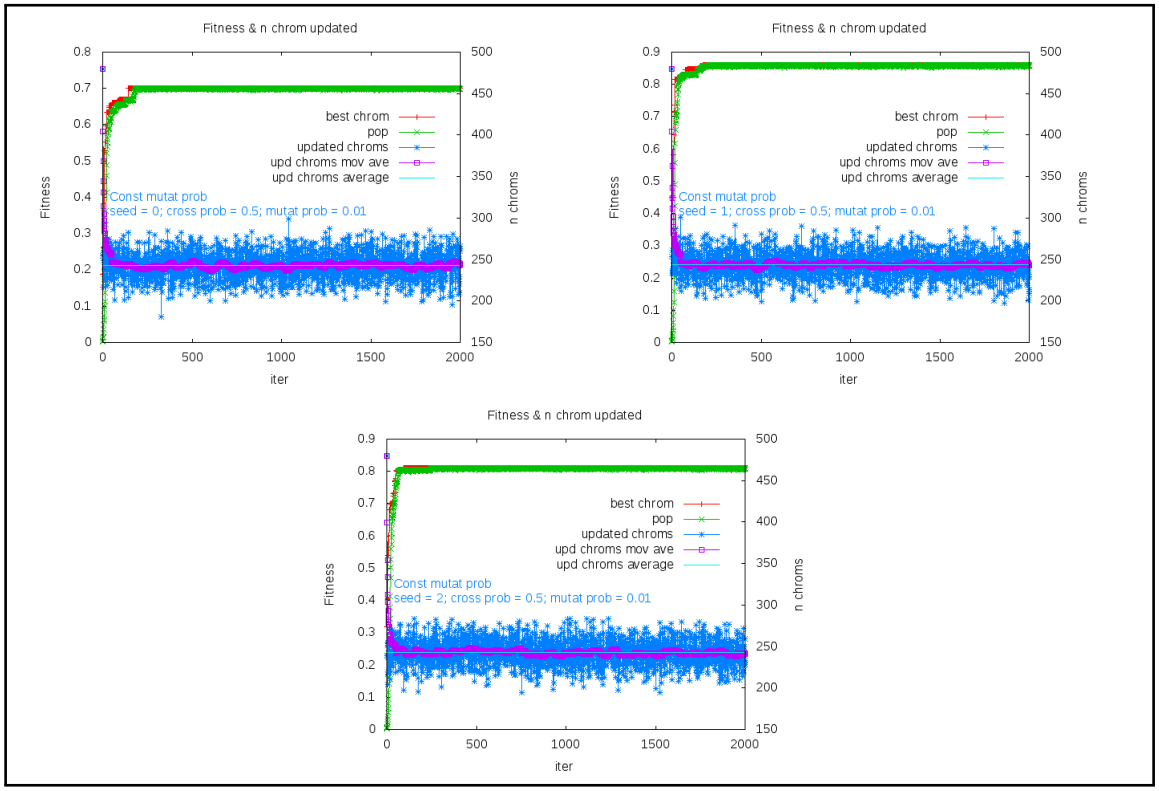


Figure A.13: Constant mutation probability 0.01

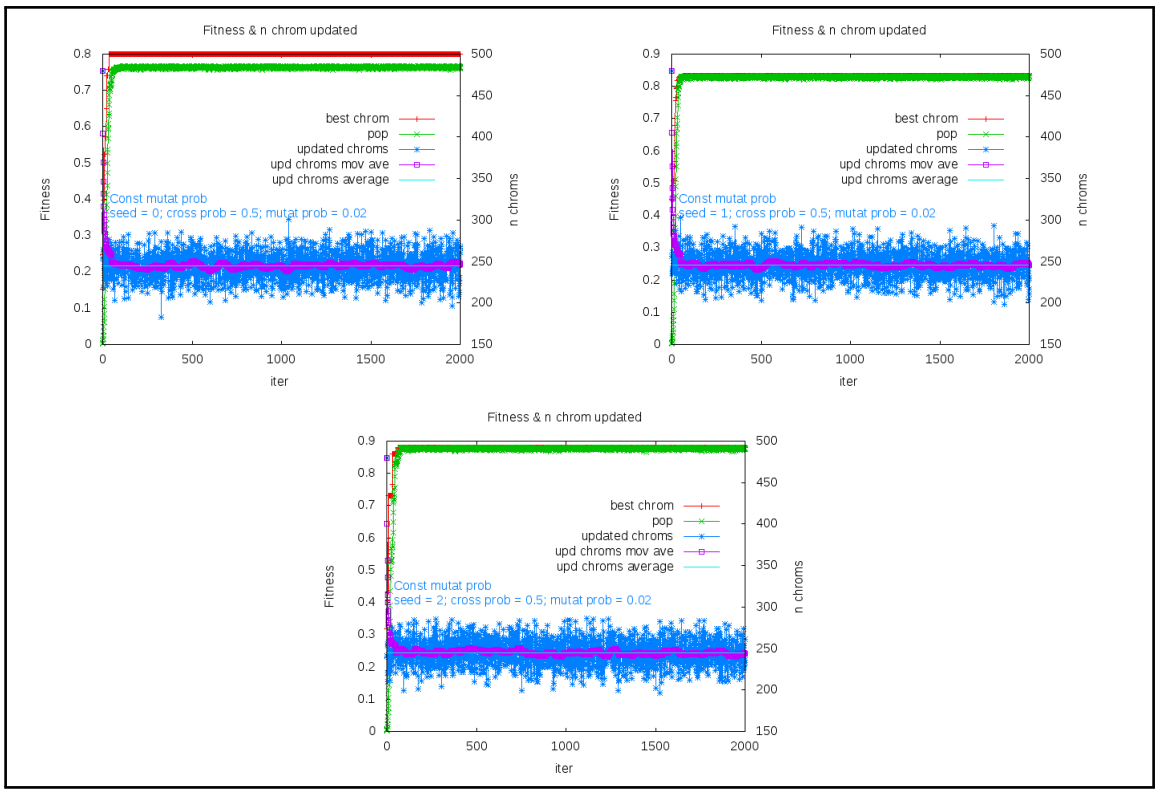


Figure A.14: Constant mutation probability 0.02

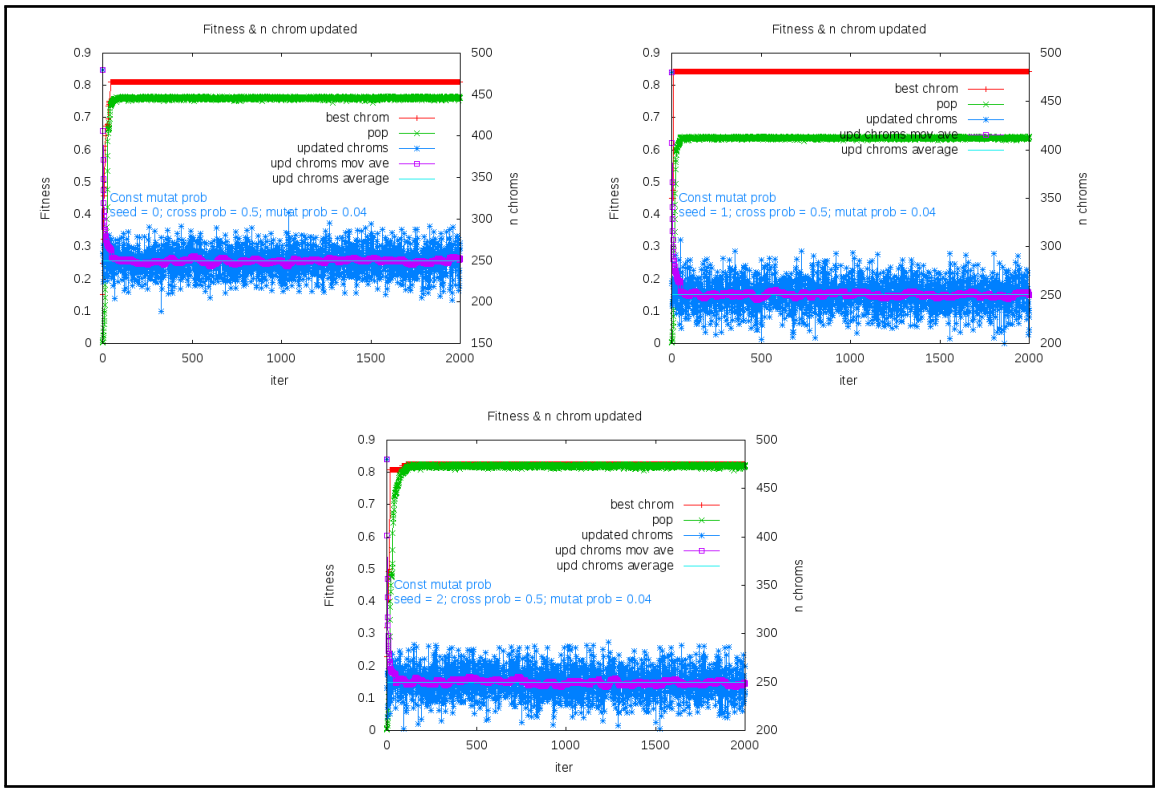


Figure A.15: Constant mutation probability 0.04

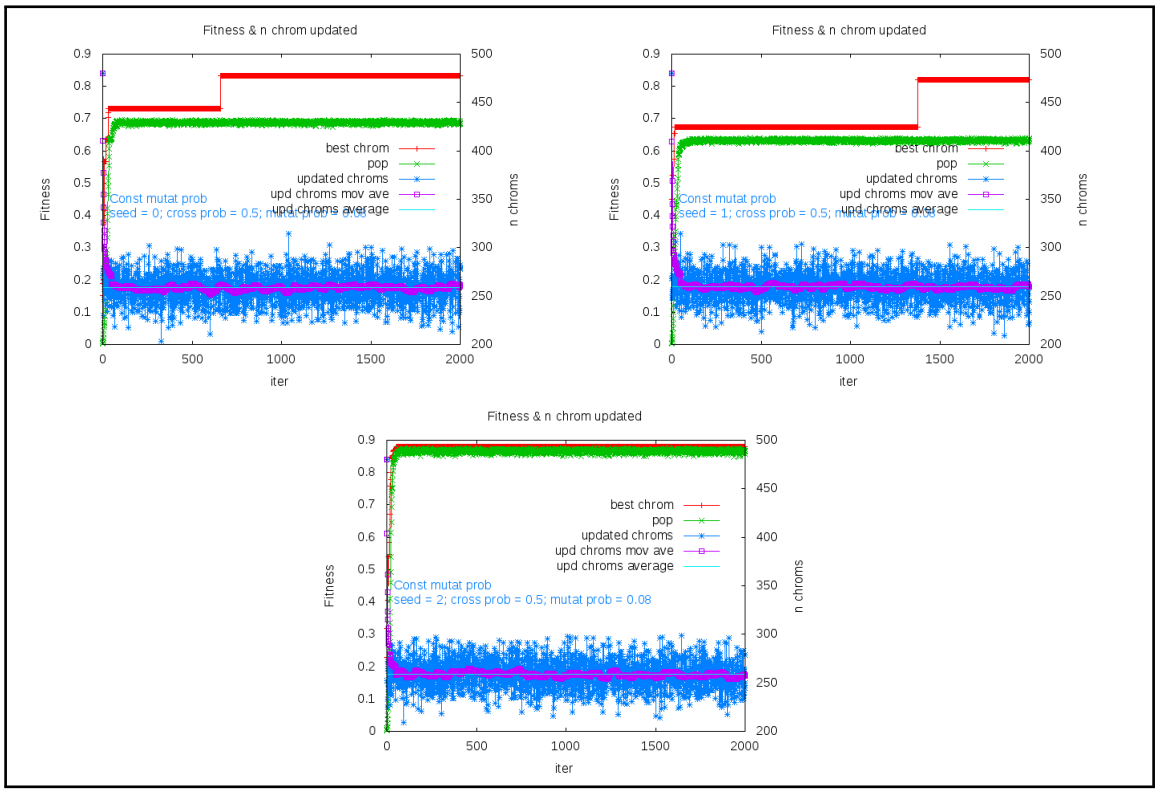


Figure A.16: Constant mutation probability 0.08

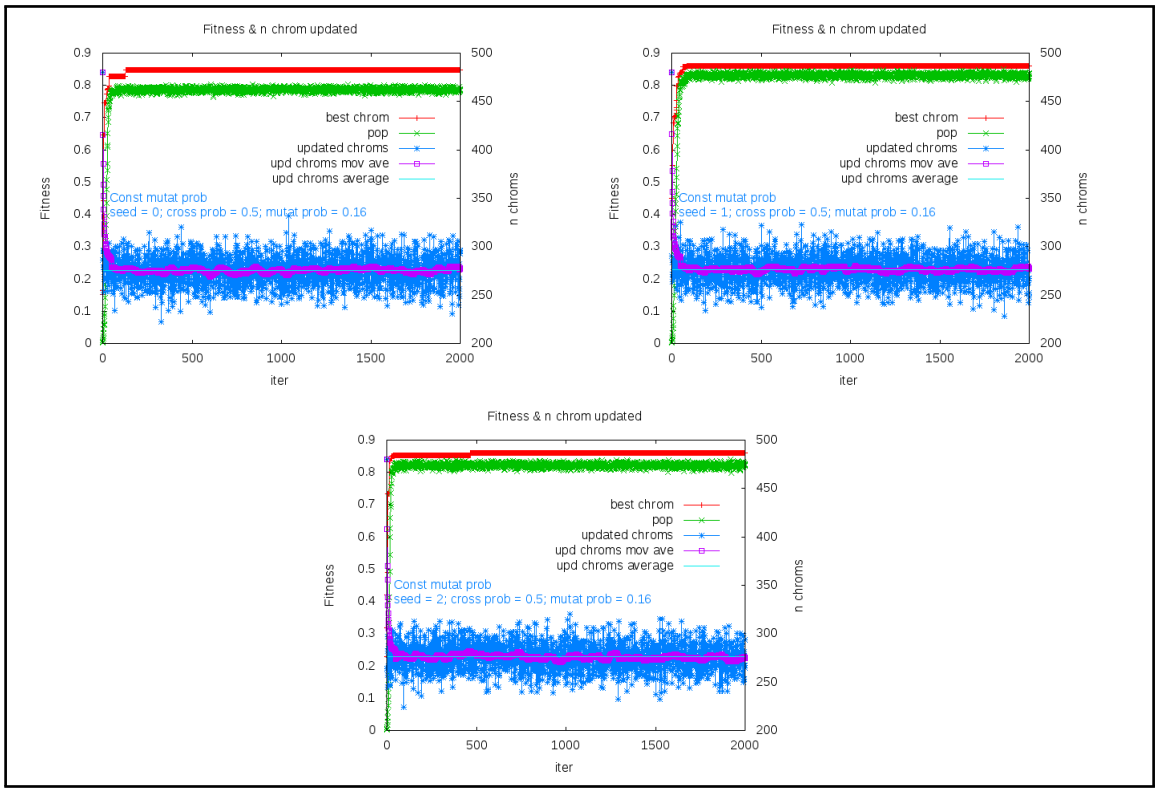


Figure A.17: Constant mutation probability 0.16

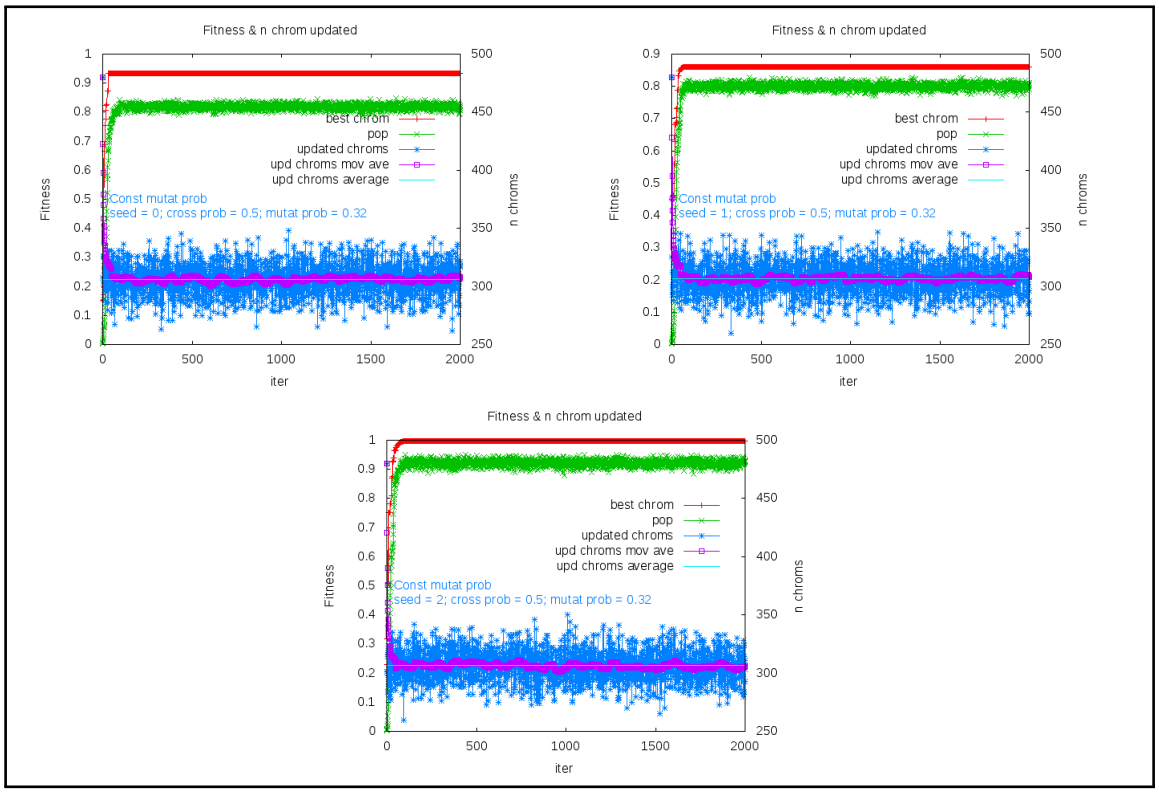


Figure A.18: Constant mutation probability 0.32

A.1.4 Crossover probability 0.6

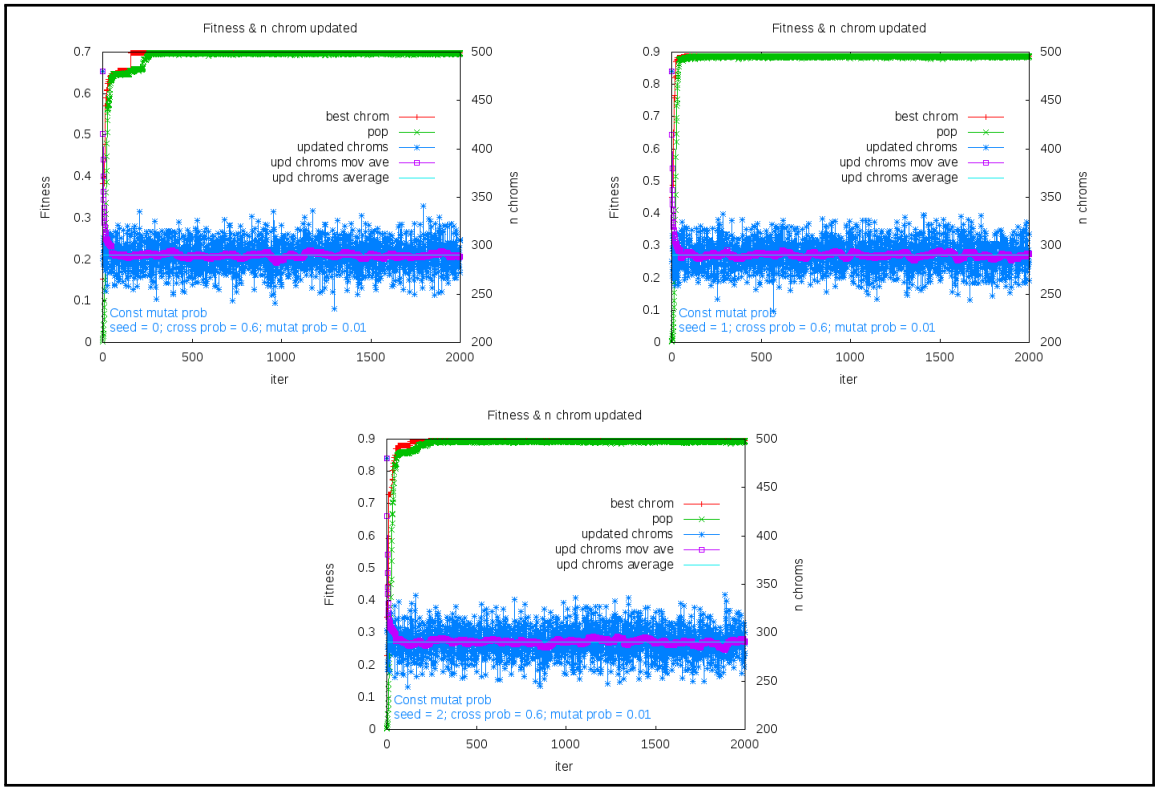


Figure A.19: Constant mutation probability 0.01

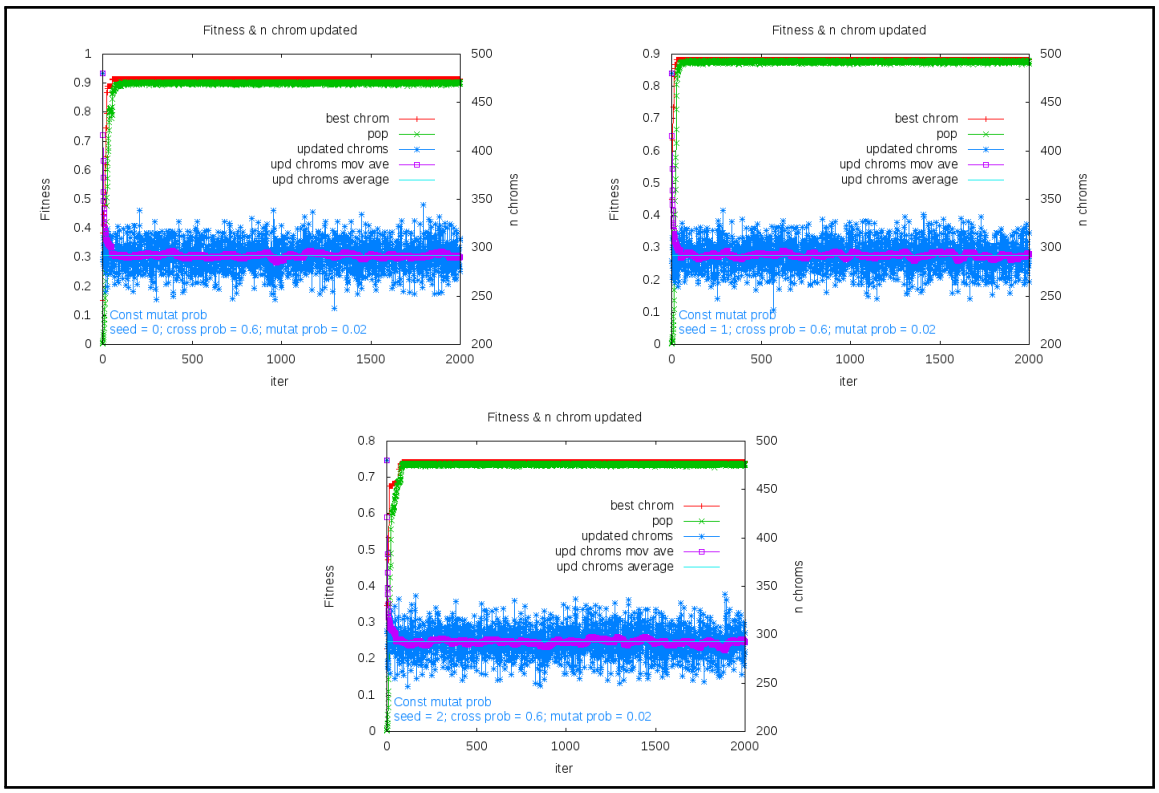


Figure A.20: Constant mutation probability 0.02

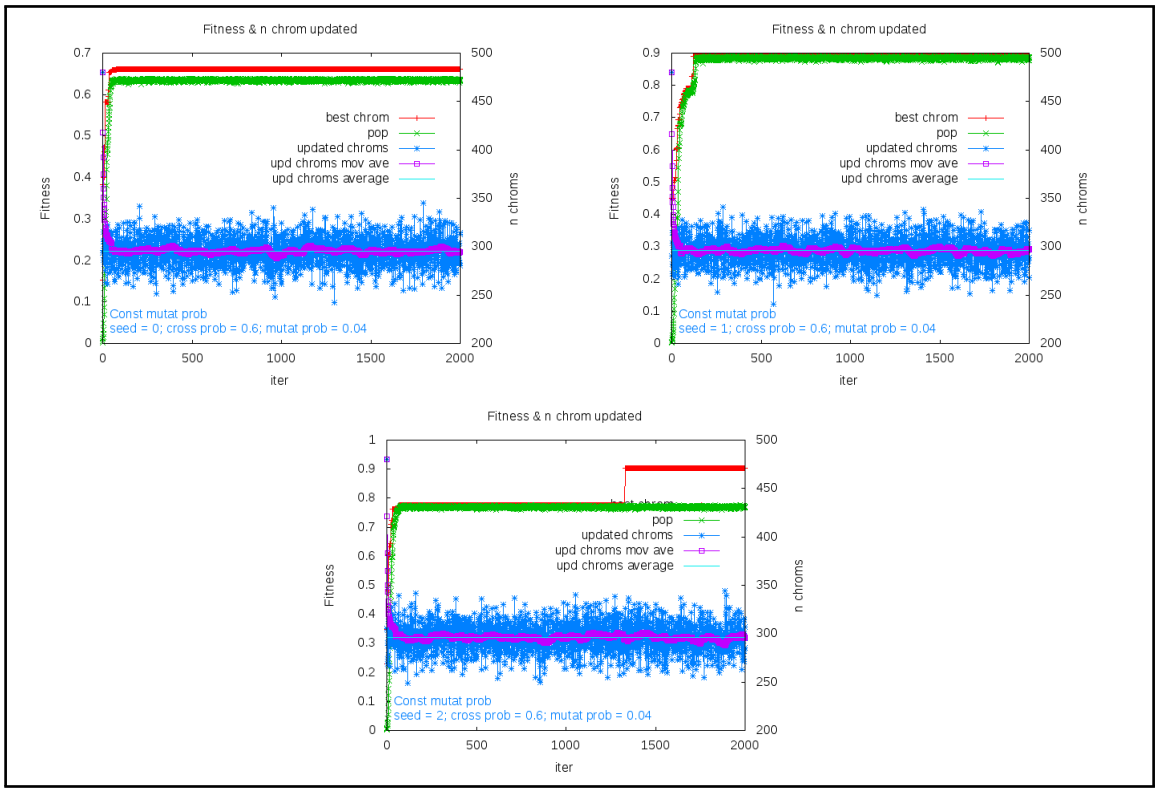


Figure A.21: Constant mutation probability 0.04

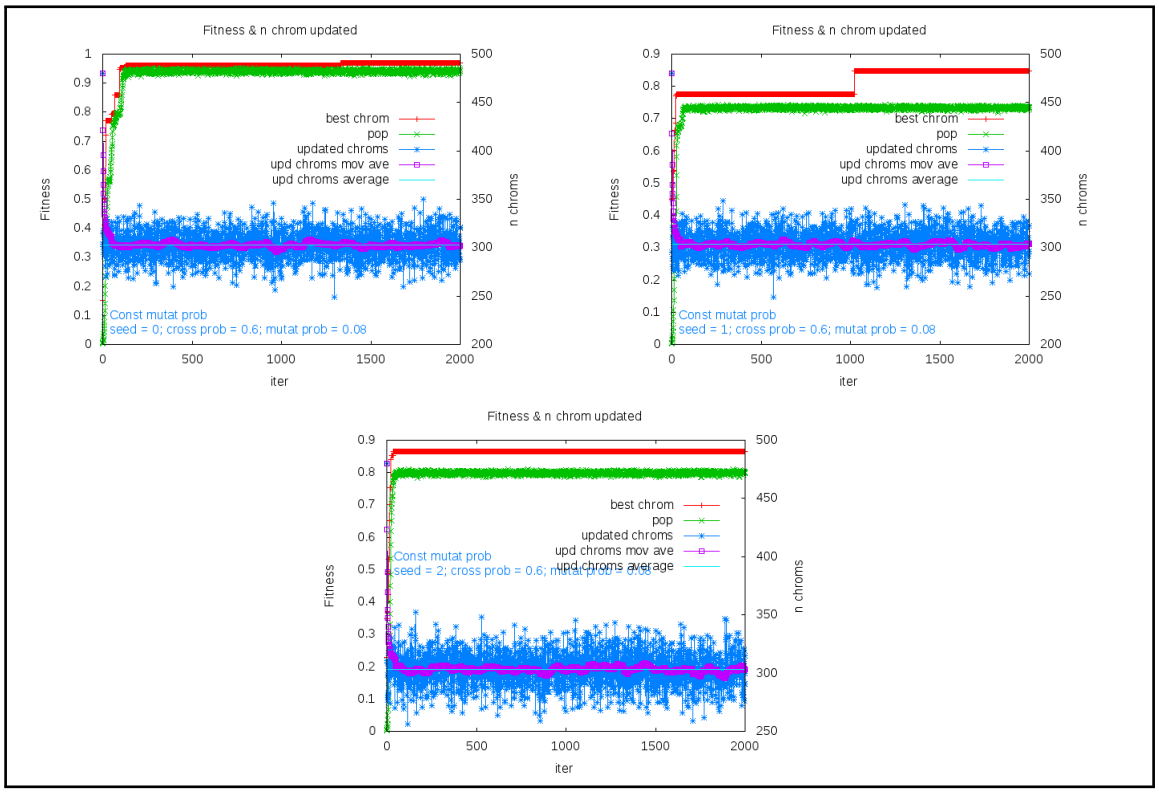


Figure A.22: Constant mutation probability 0.08

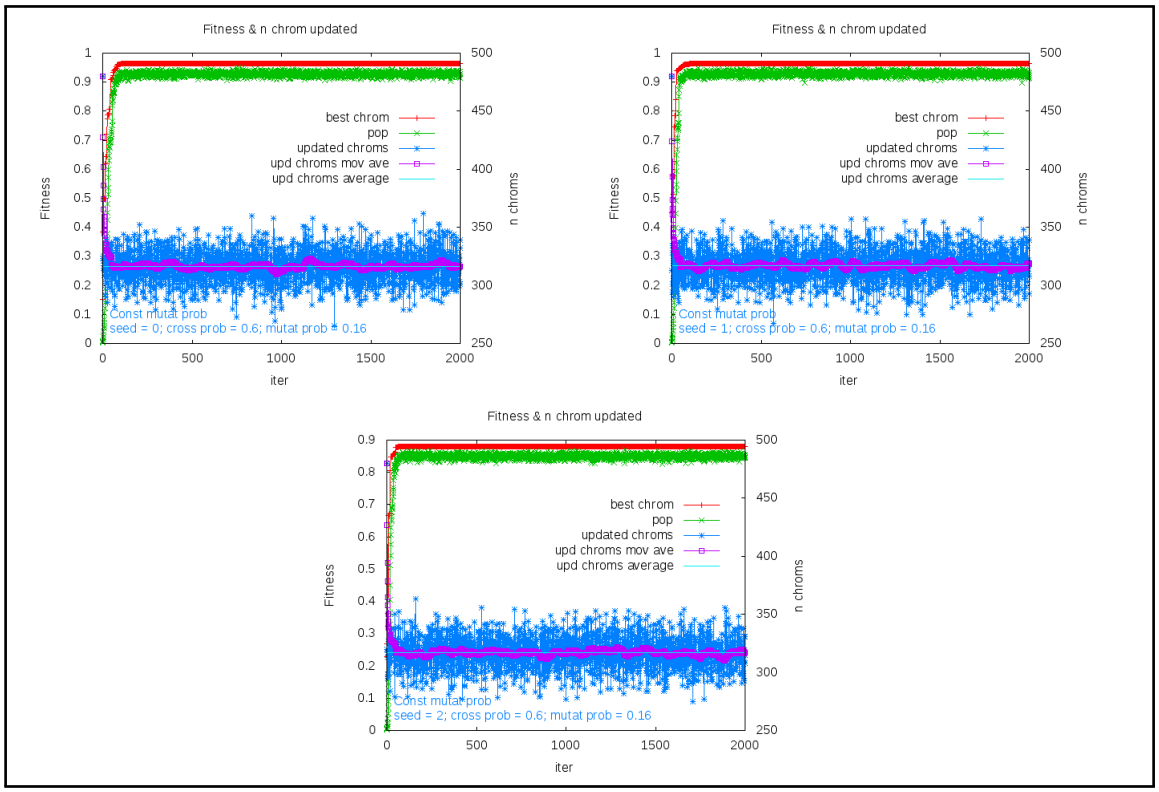


Figure A.23: Constant mutation probability 0.16

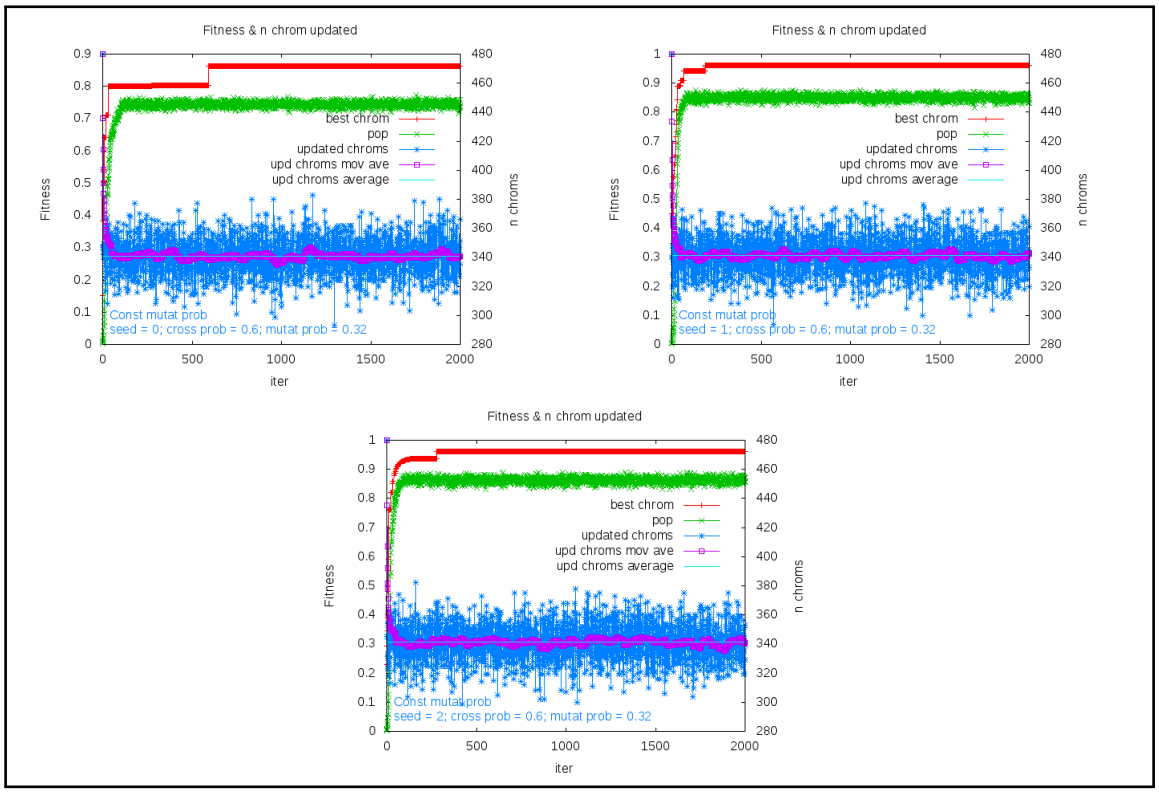


Figure A.24: Constant mutation probability 0.32

A.1.5 Crossover probability 0.7

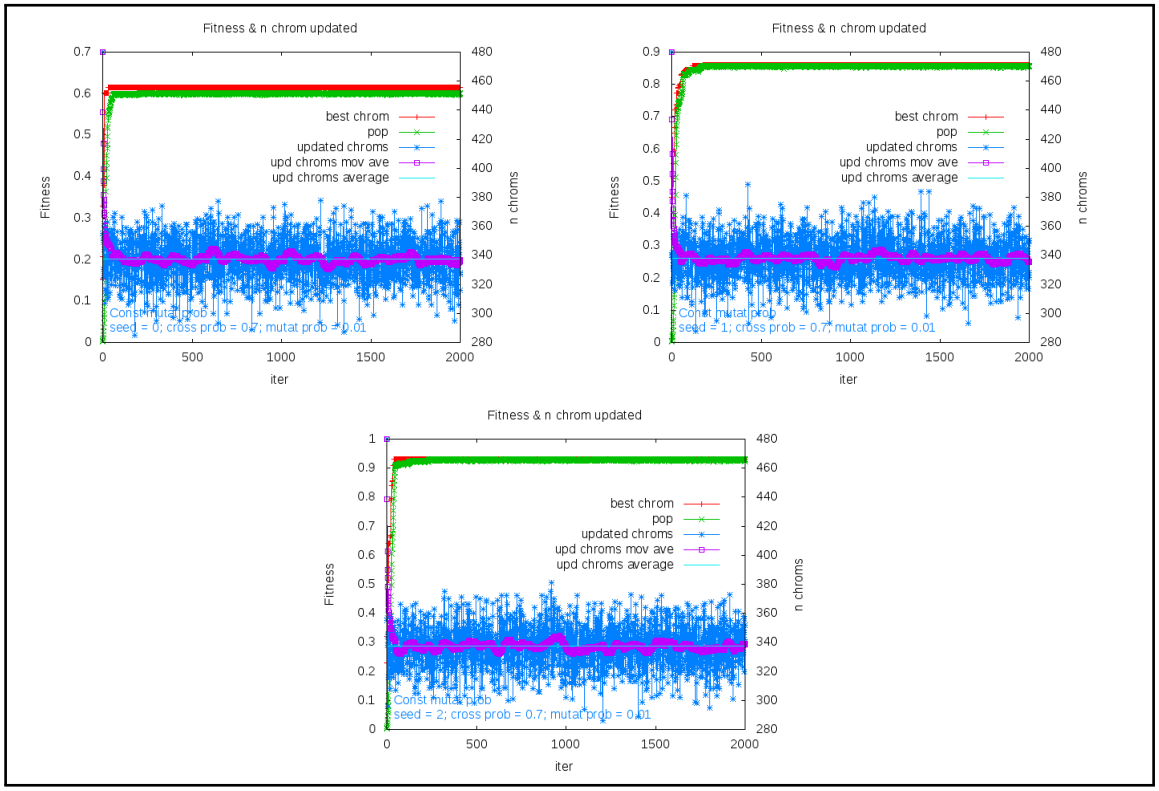


Figure A.25: Constant mutation probability 0.01

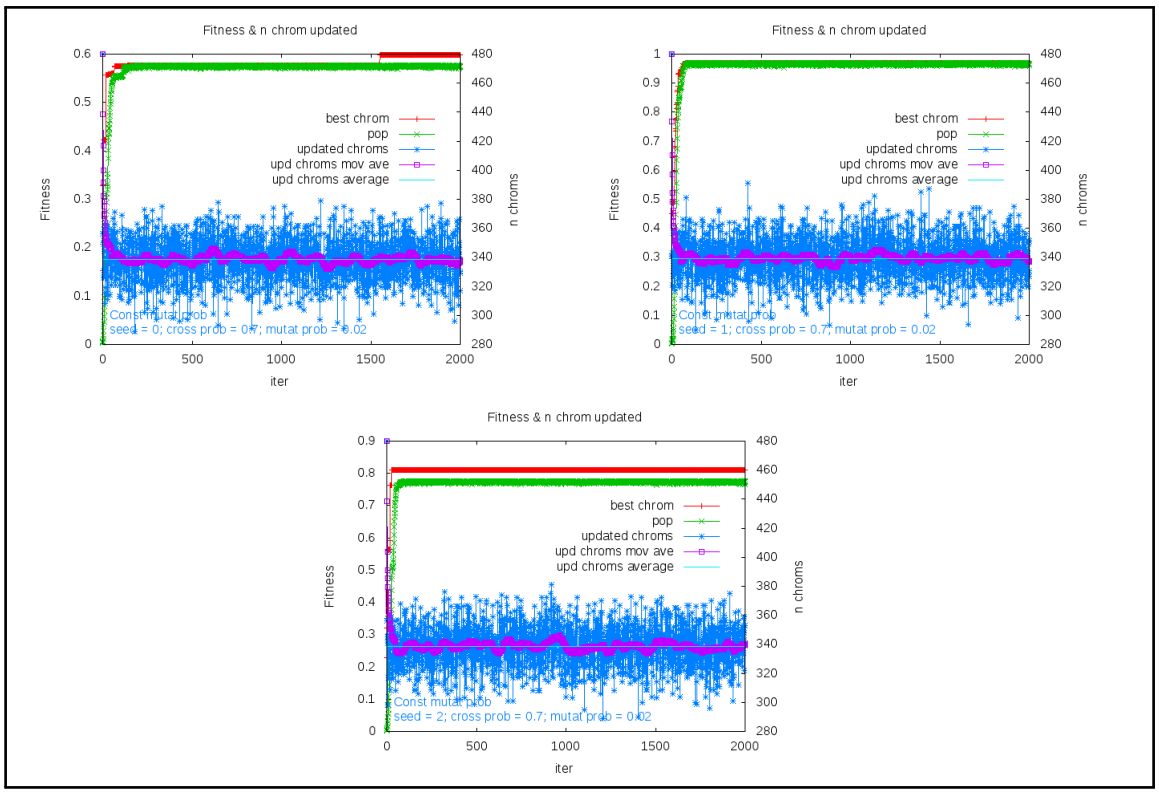


Figure A.26: Constant mutation probability 0.02

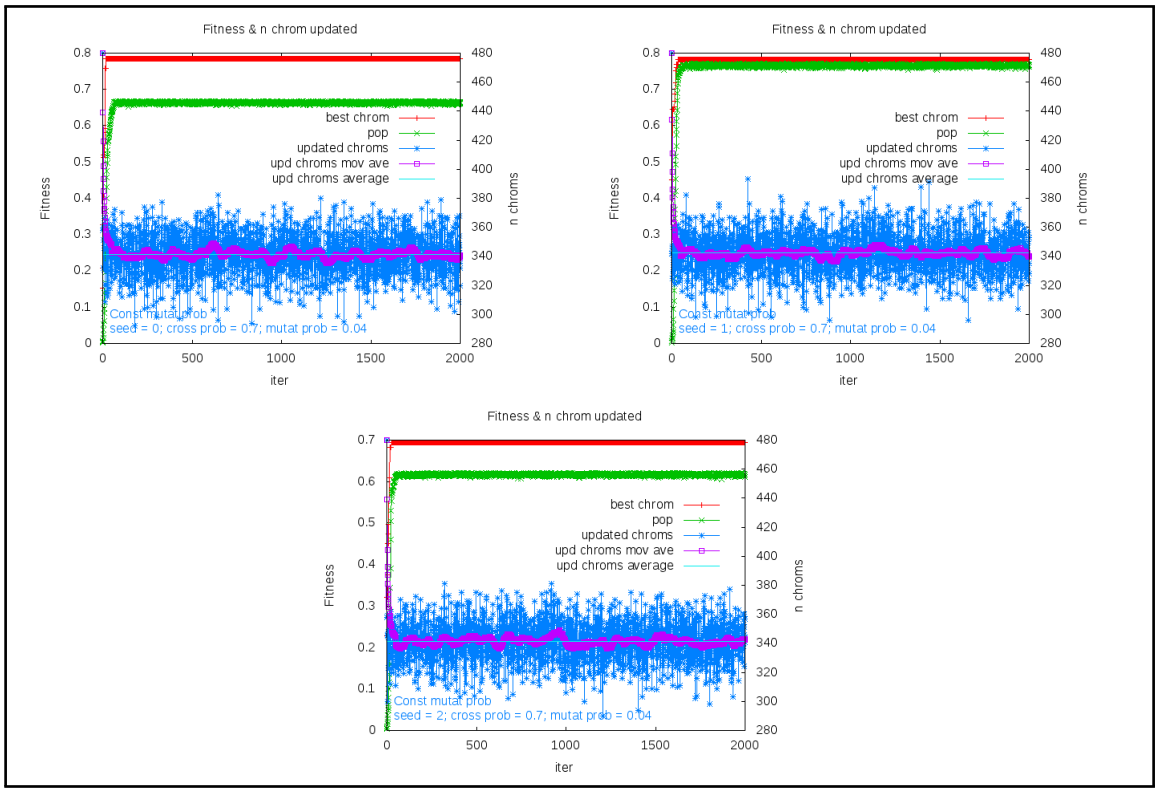


Figure A.27: Constant mutation probability 0.04

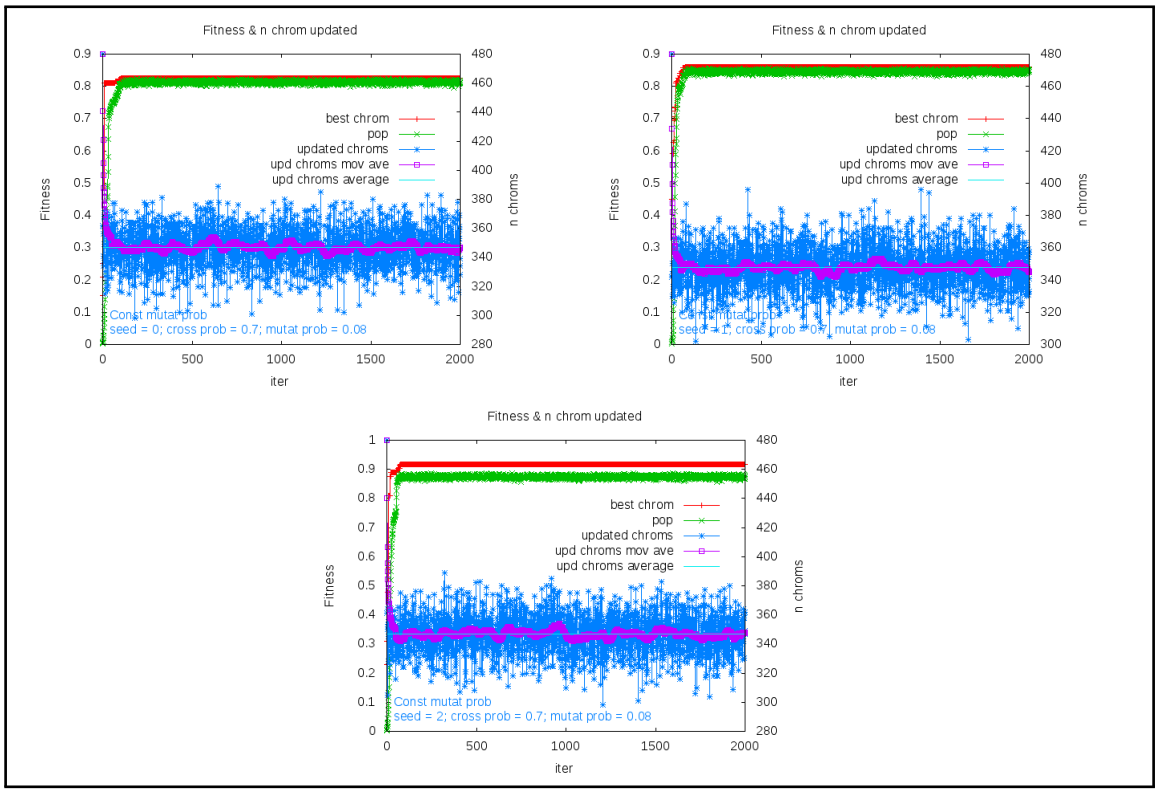


Figure A.28: Constant mutation probability 0.08

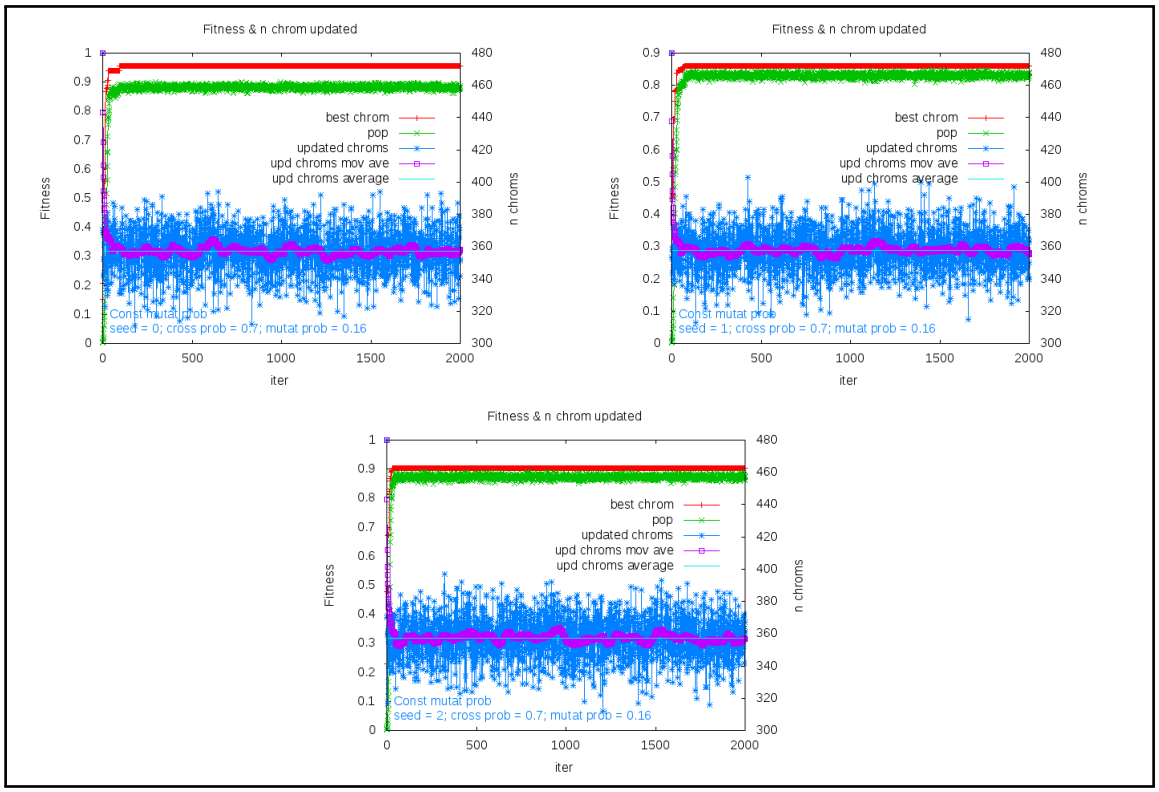


Figure A.29: Constant mutation probability 0.16

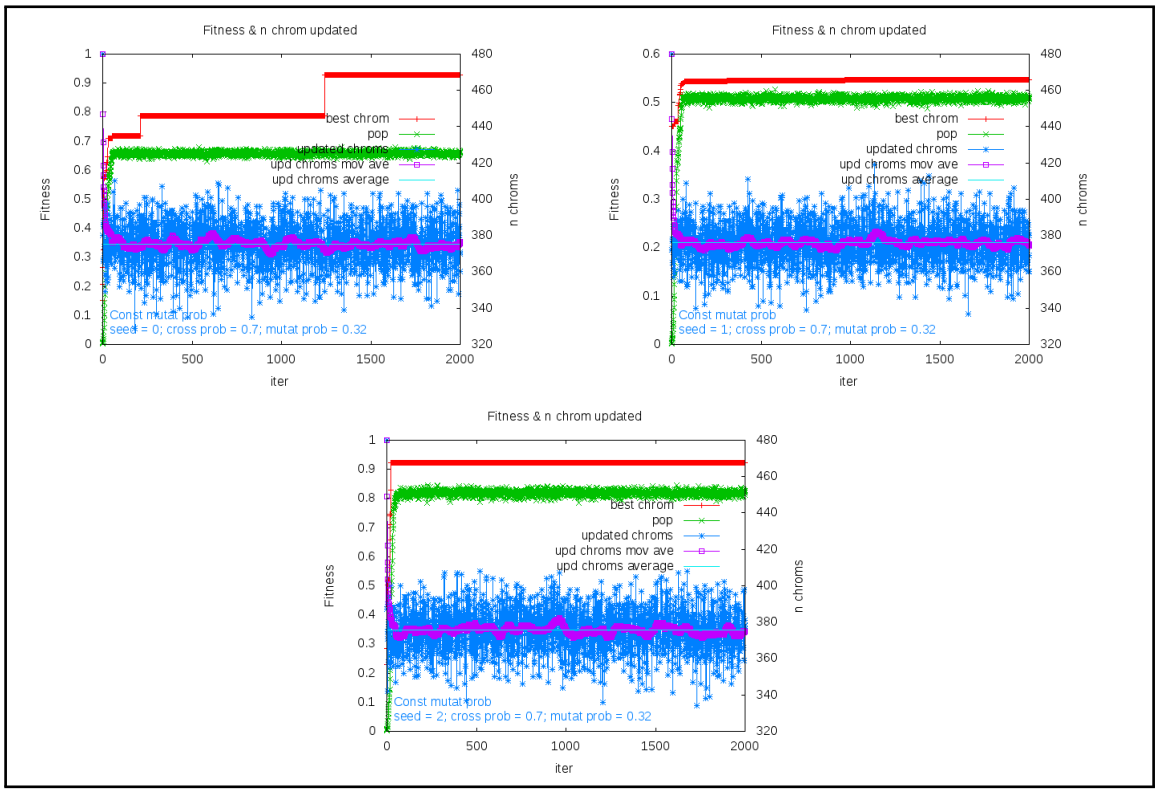


Figure A.30: Constant mutation probability 0.32

A.1.6 Crossover probability 0.8

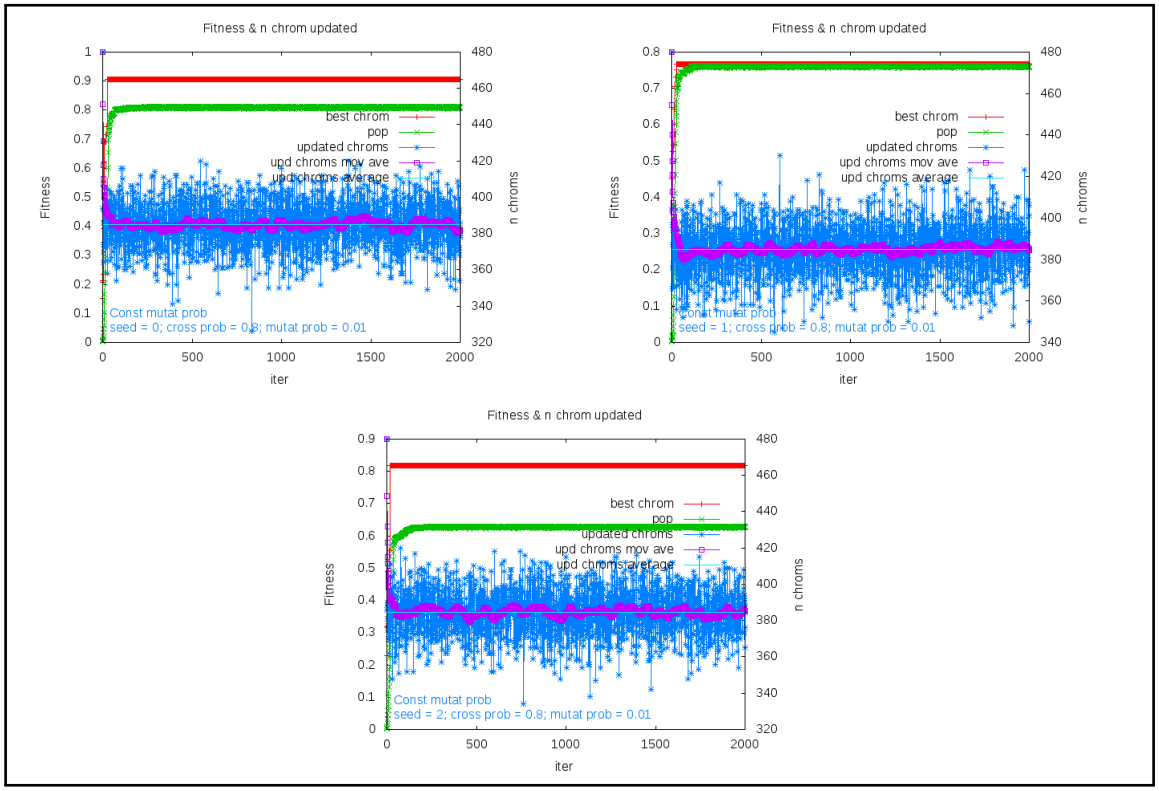


Figure A.31: Constant mutation probability 0.01

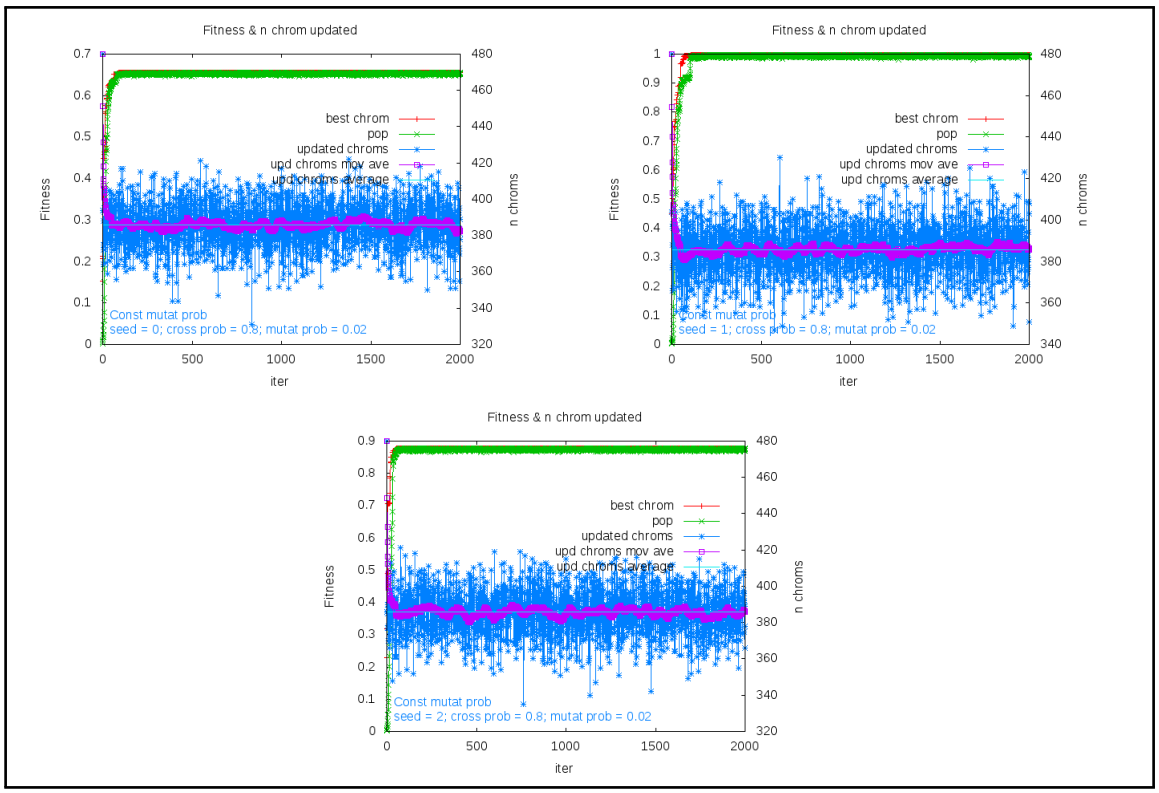


Figure A.32: Constant mutation probability 0.02

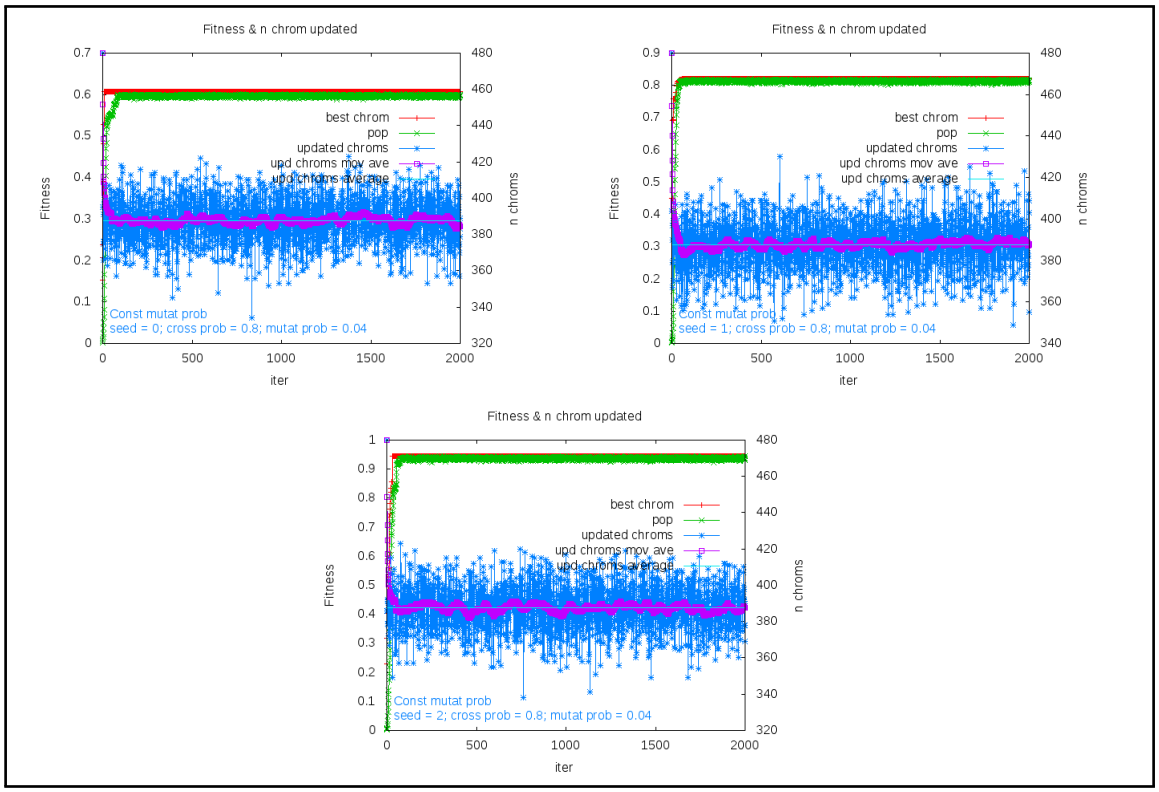


Figure A.33: Constant mutation probability 0.04

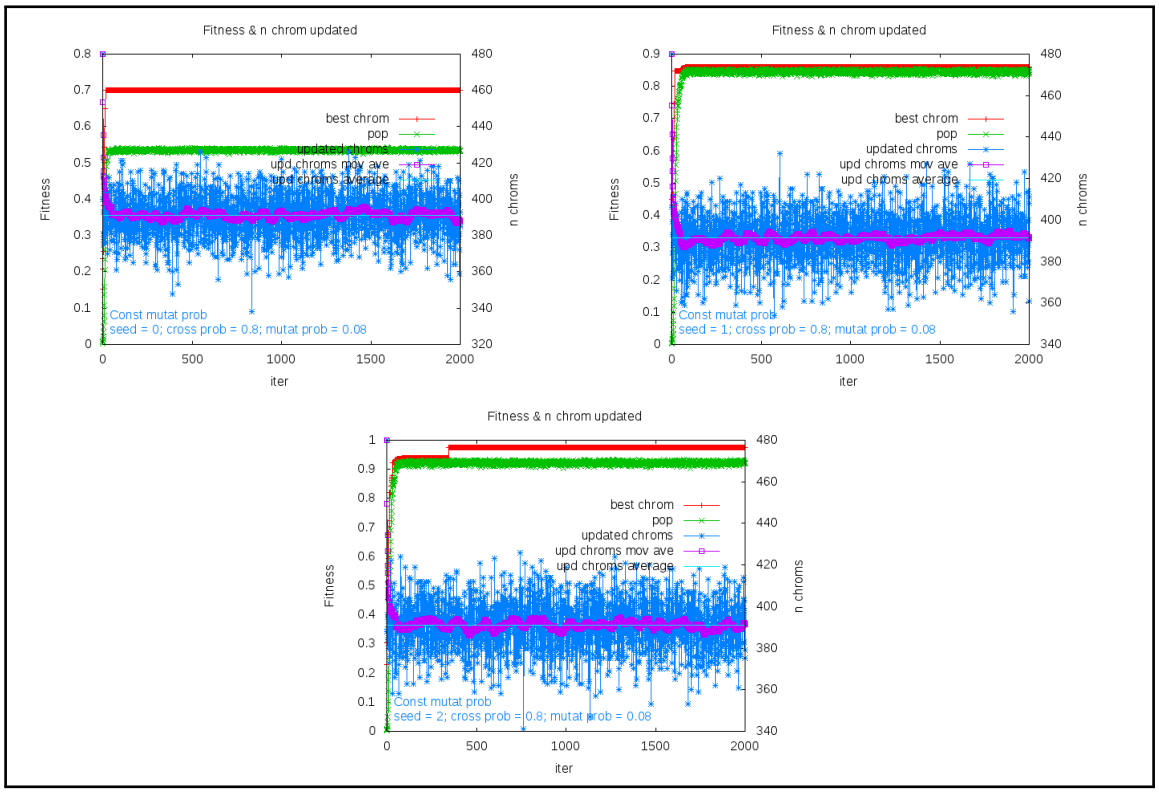


Figure A.34: Constant mutation probability 0.08

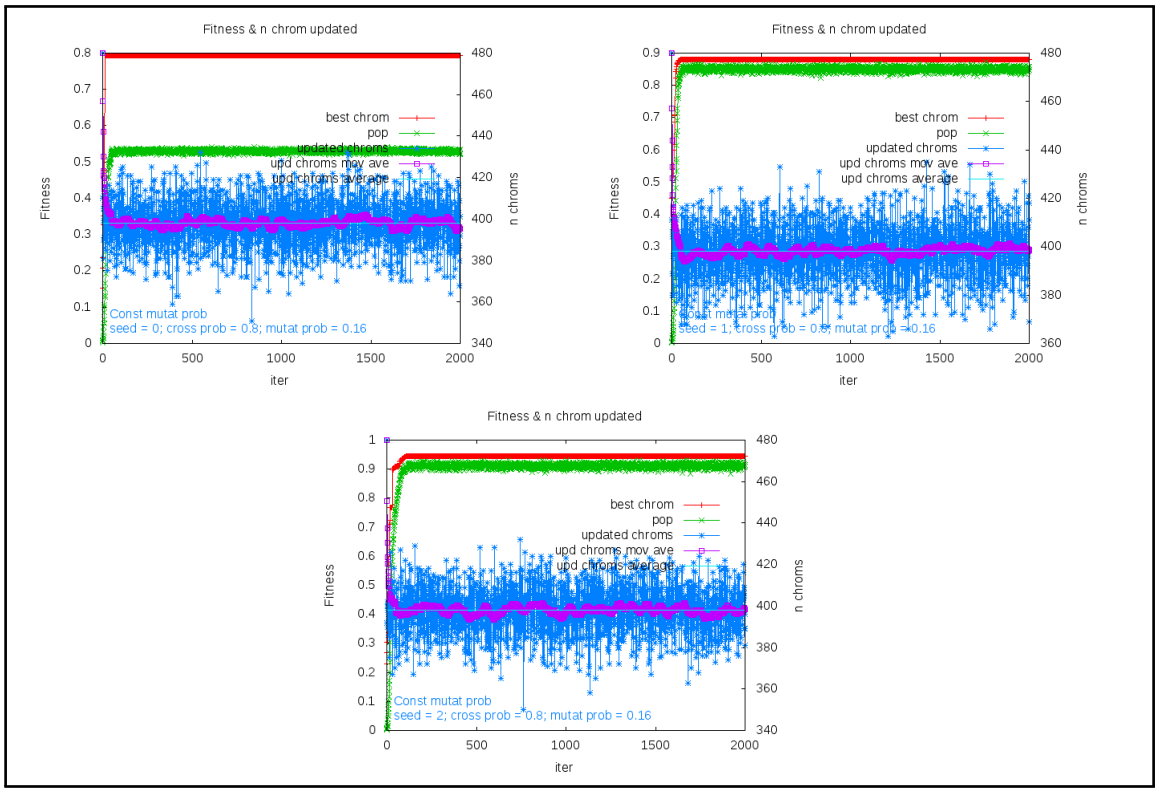


Figure A.35: Constant mutation probability 0.16

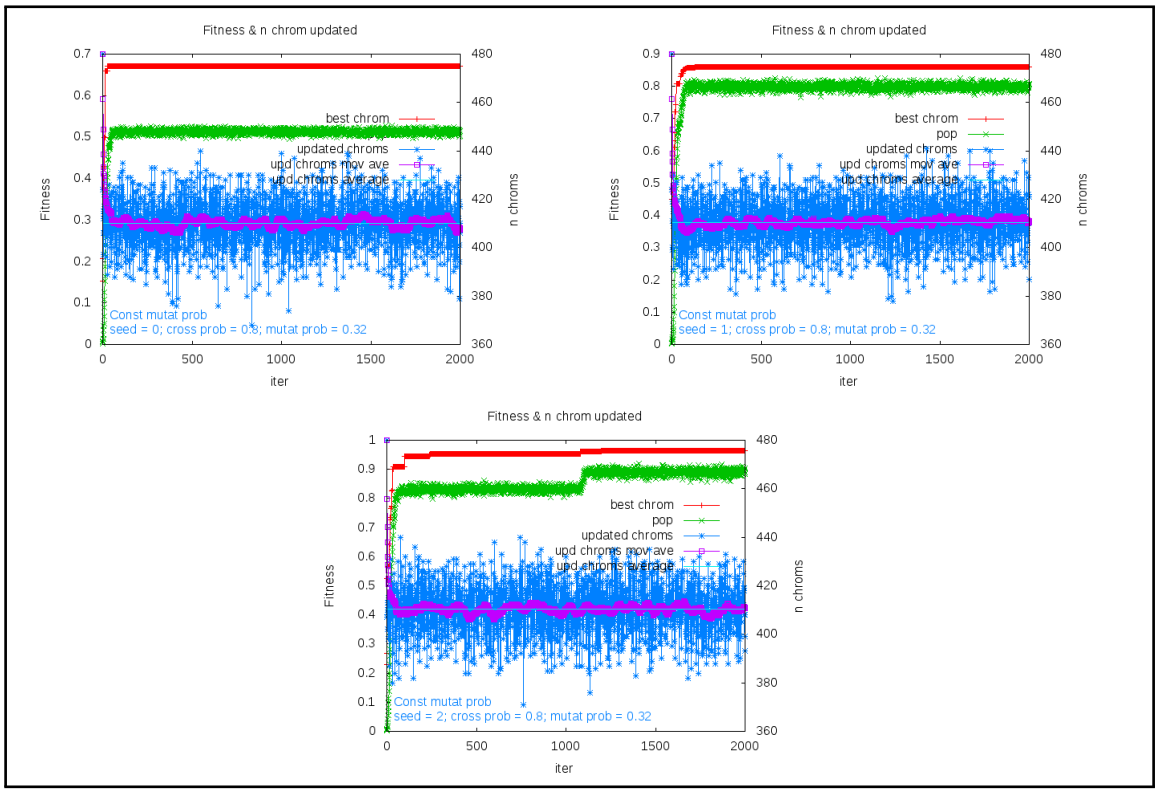


Figure A.36: Constant mutation probability 0.32

A.1.7 Crossover probability 0.9

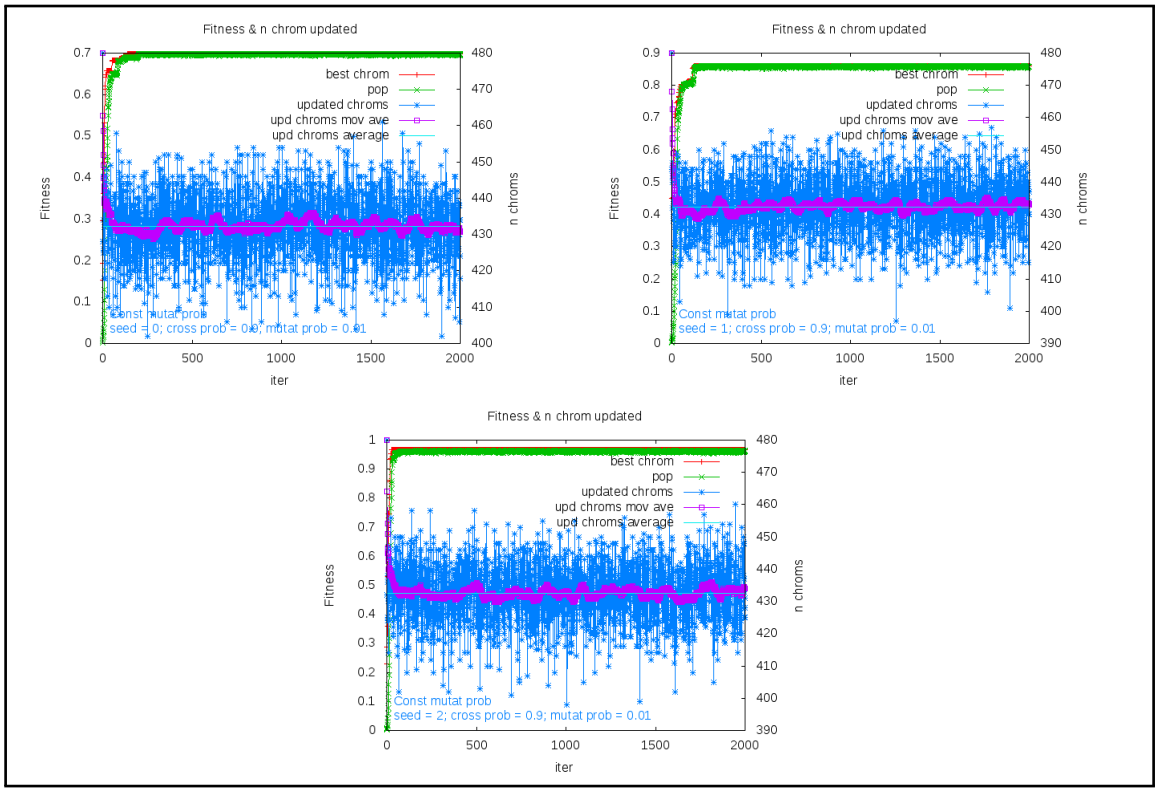


Figure A.37: Constant mutation probability 0.01

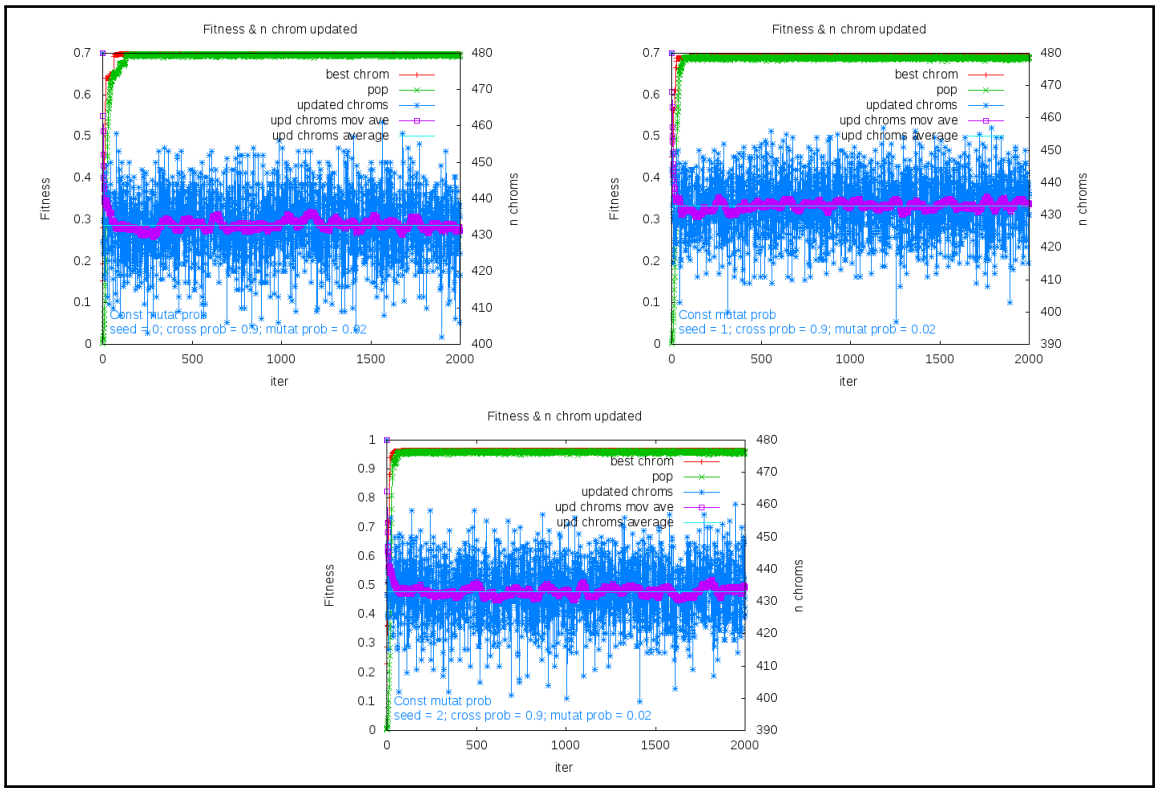


Figure A.38: Constant mutation probability 0.02

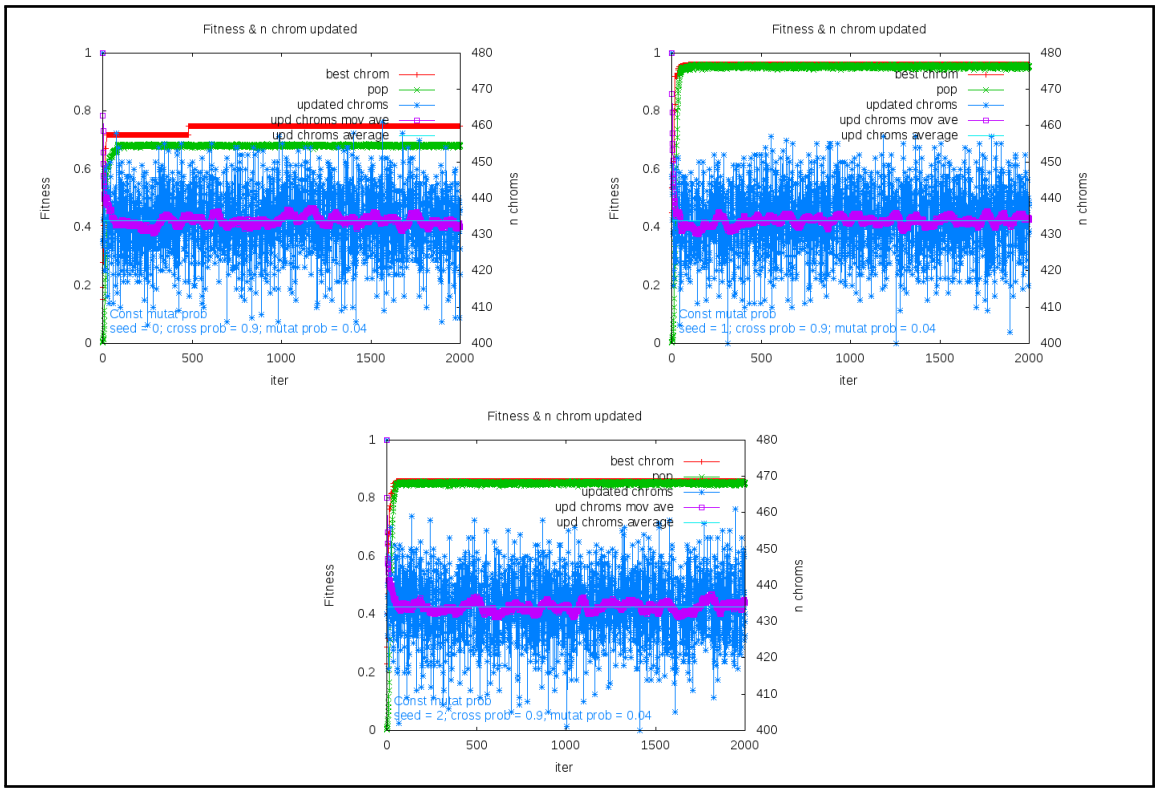


Figure A.39: Constant mutation probability 0.04

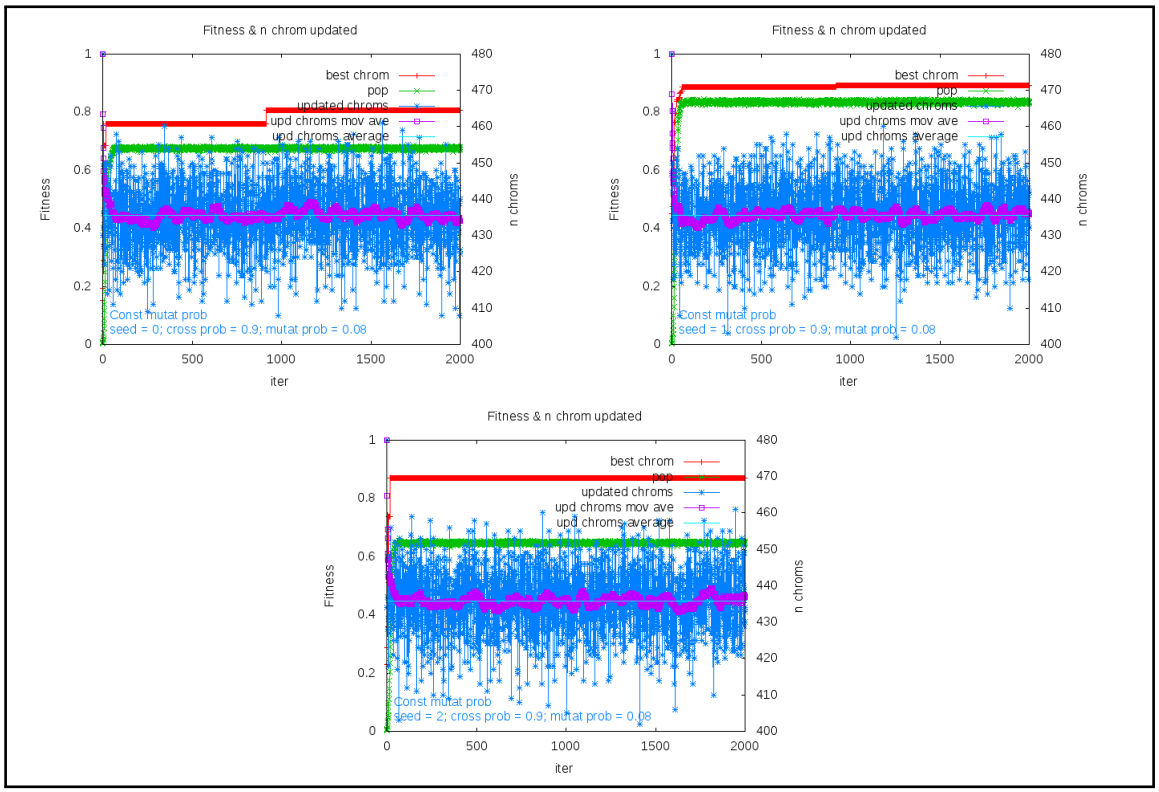


Figure A.40: Constant mutation probability 0.08

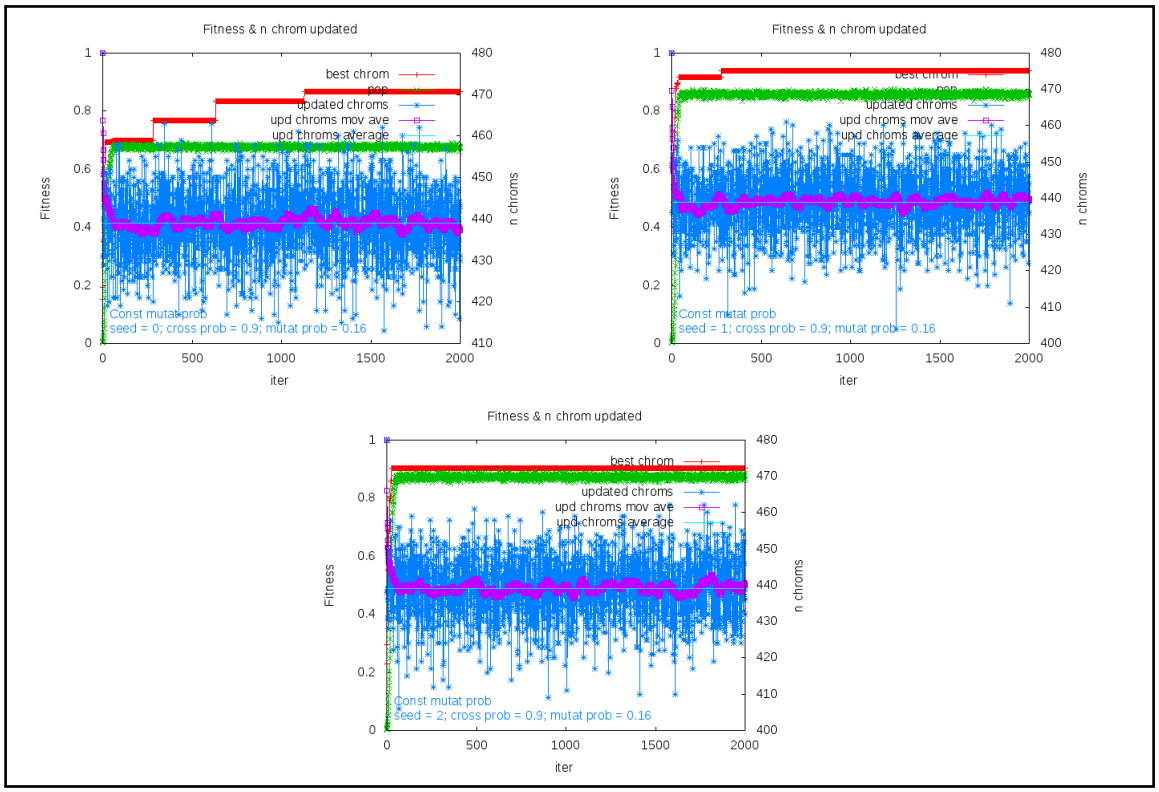


Figure A.41: Constant mutation probability 0.16

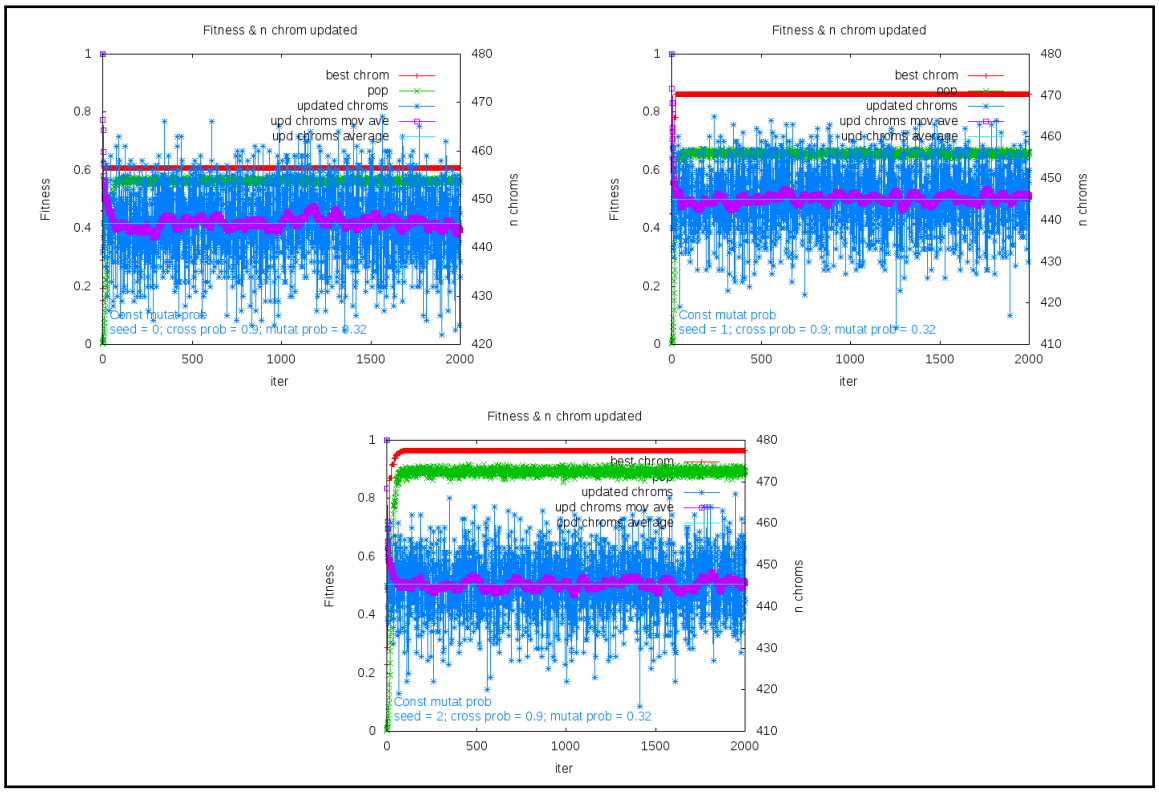


Figure A.42: Constant mutation probability 0.32

A.2 Variable mutation probability test set

The set of tests in which the mutation probability is linearly decreased during evolution of the GA produced the graphs shown in sections A.2.1 to A.2.7, which results from sequentially scanning the values in eq. A.1. Each section contains the graphs for all mutation probabilities in eq. A.2 but these values are now used as initial values for the mutation probability, which decreases linearly towards zero across the evolution of the GA.

A.2.1 Crossover probability 0.3

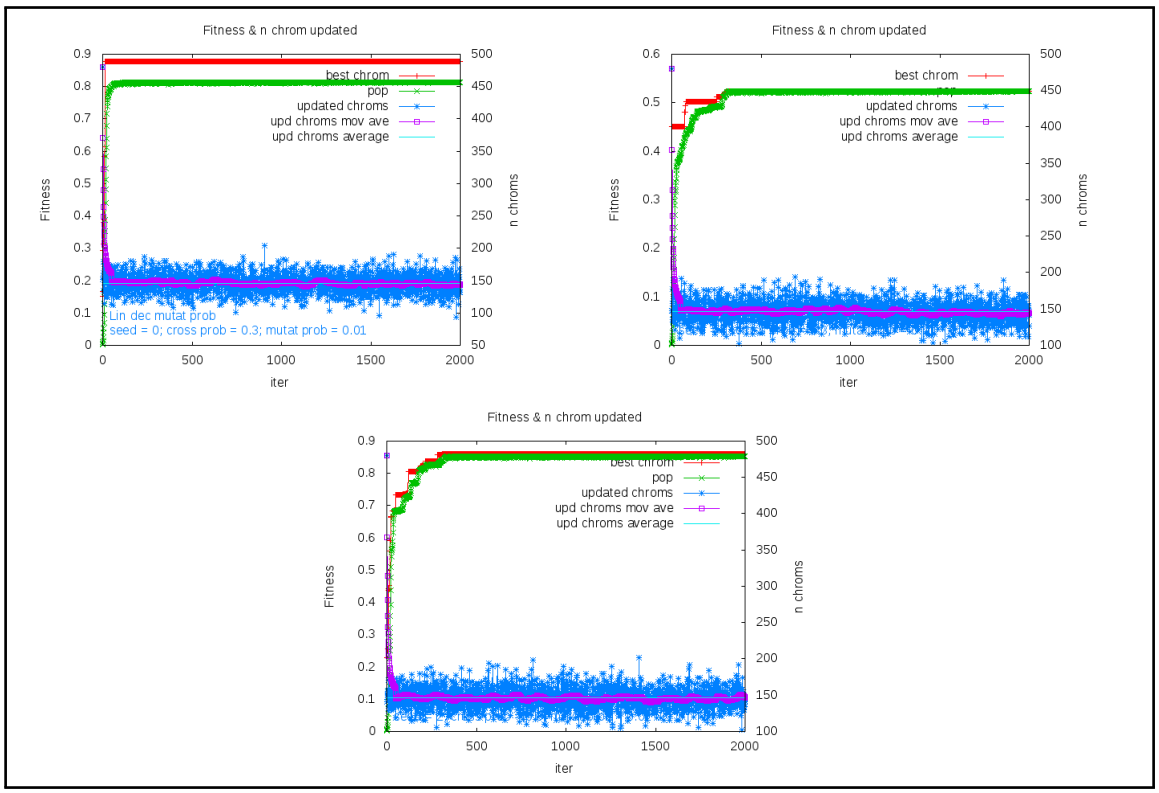


Figure A.43: Init mutation probability 0.01

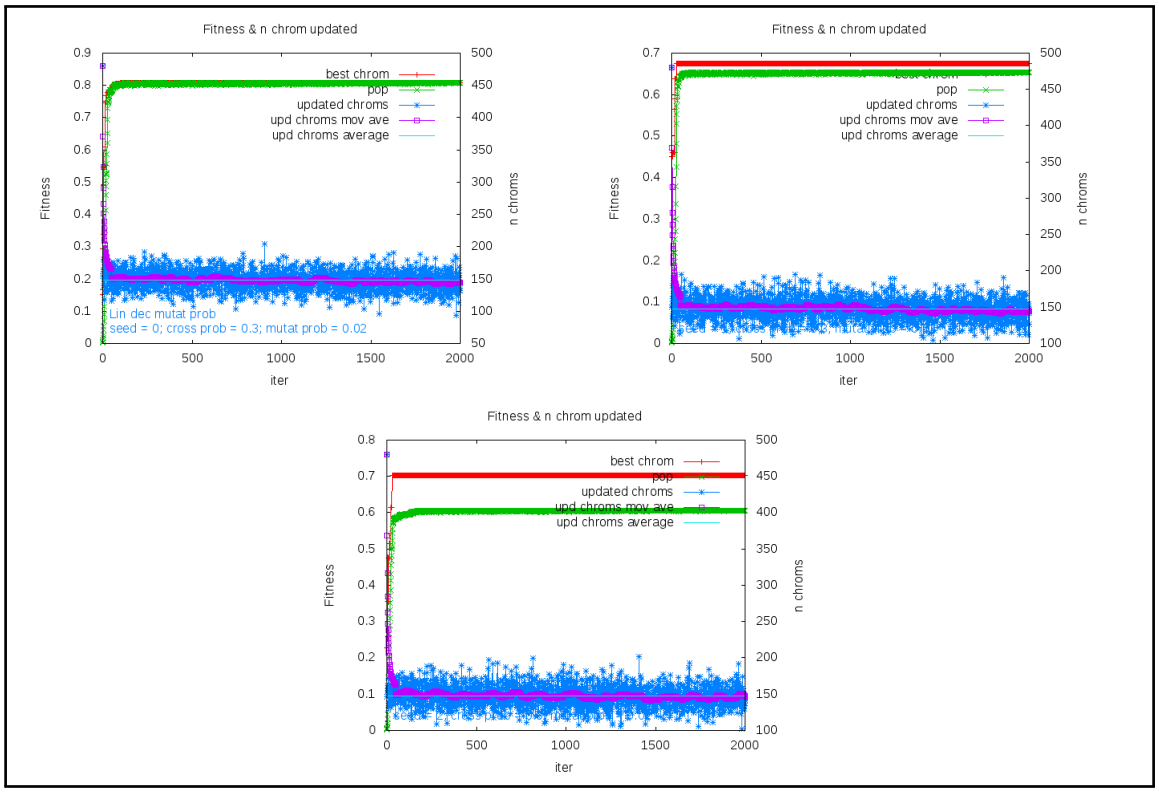


Figure A.44: Init mutation probability 0.02

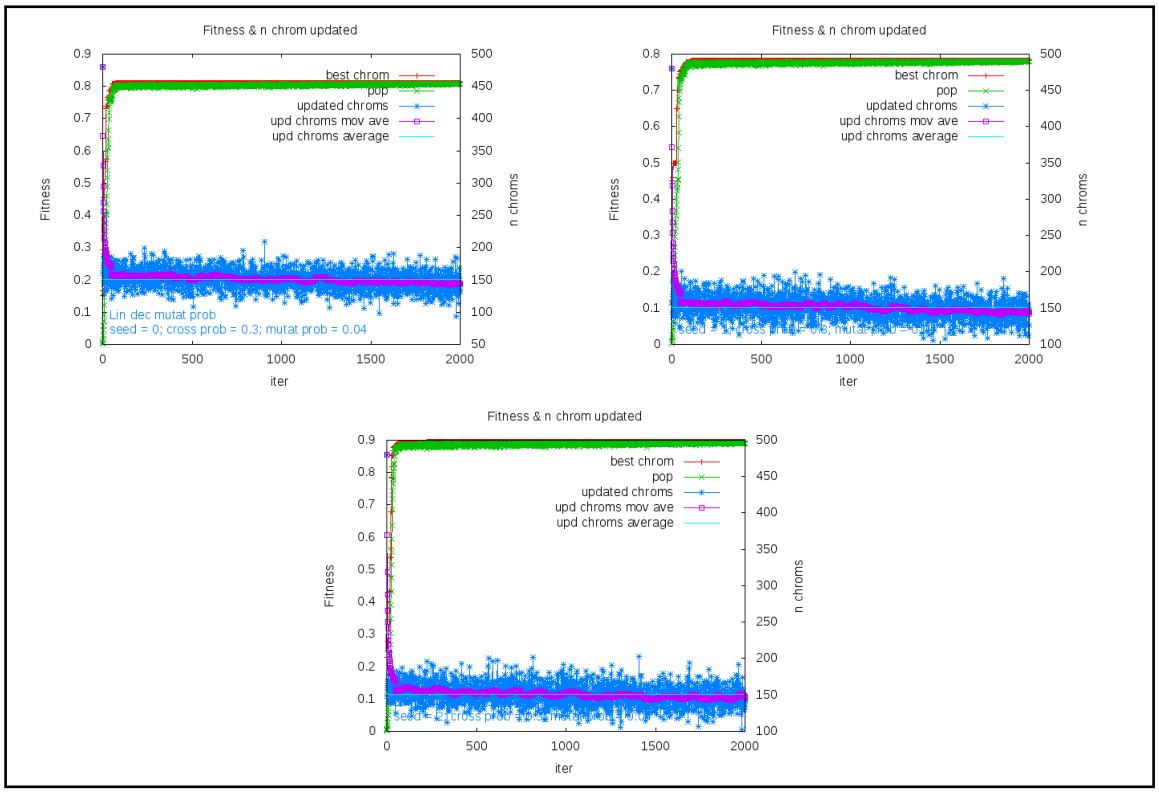


Figure A.45: Init mutation probability 0.04

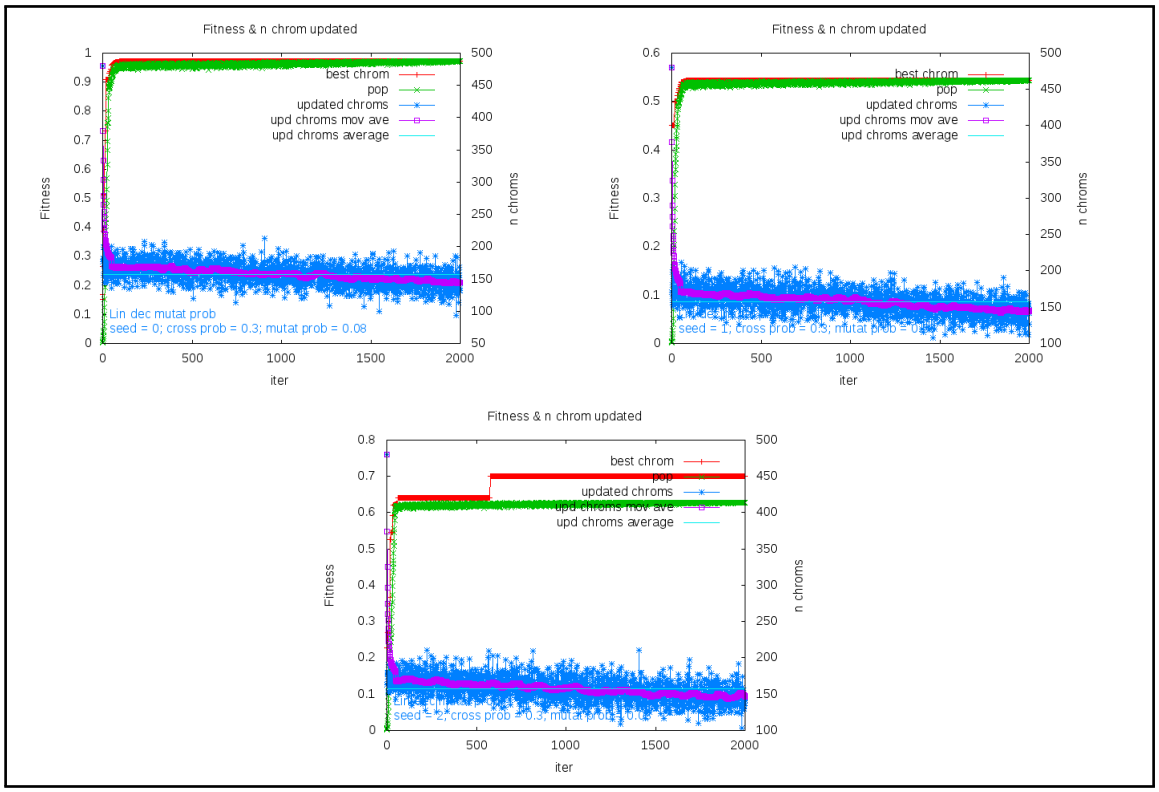


Figure A.46: Init mutation probability 0.08

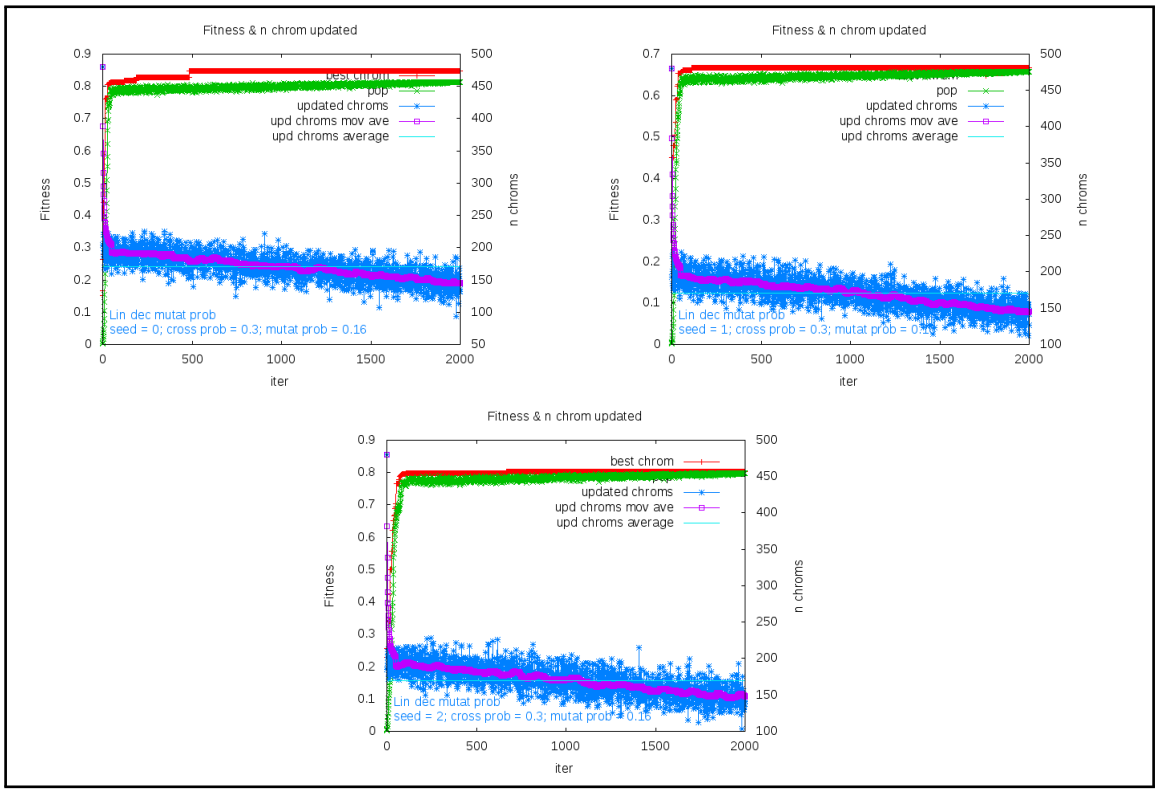


Figure A.47: Init mutation probability 0.16

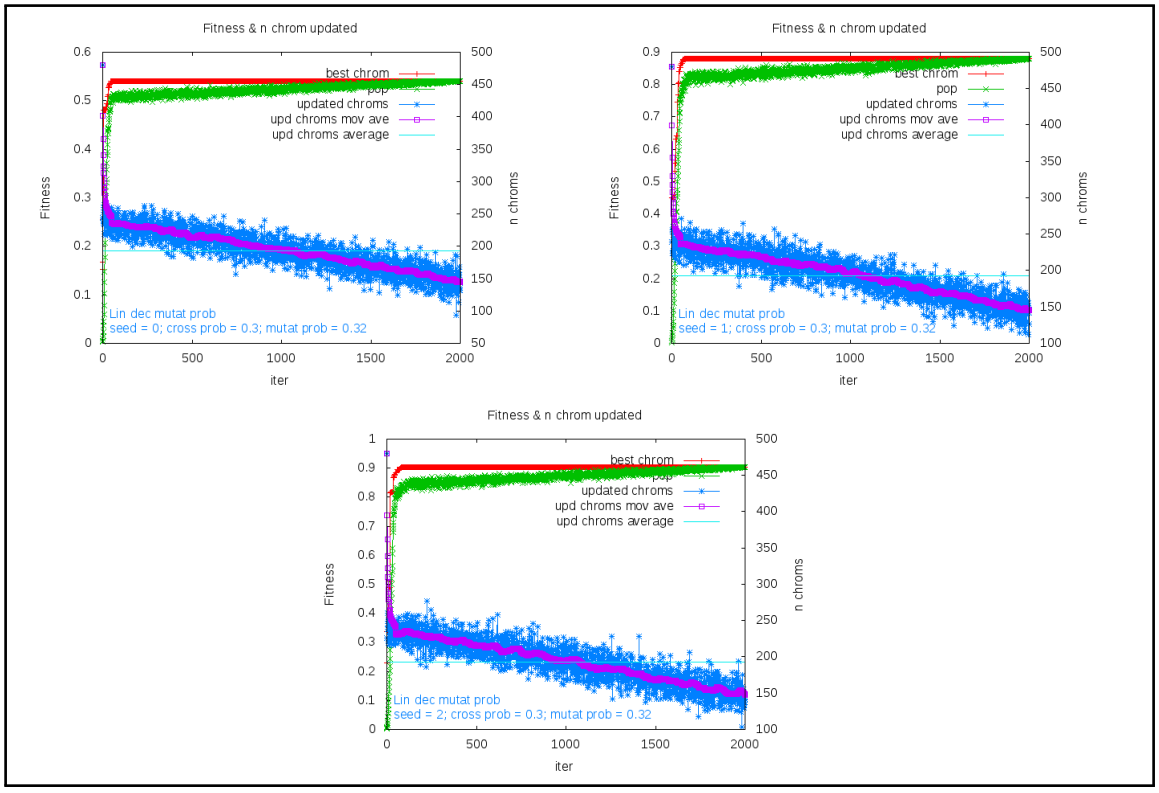


Figure A.48: Init mutation probability 0.32

A.2.2 Crossover probability 0.4

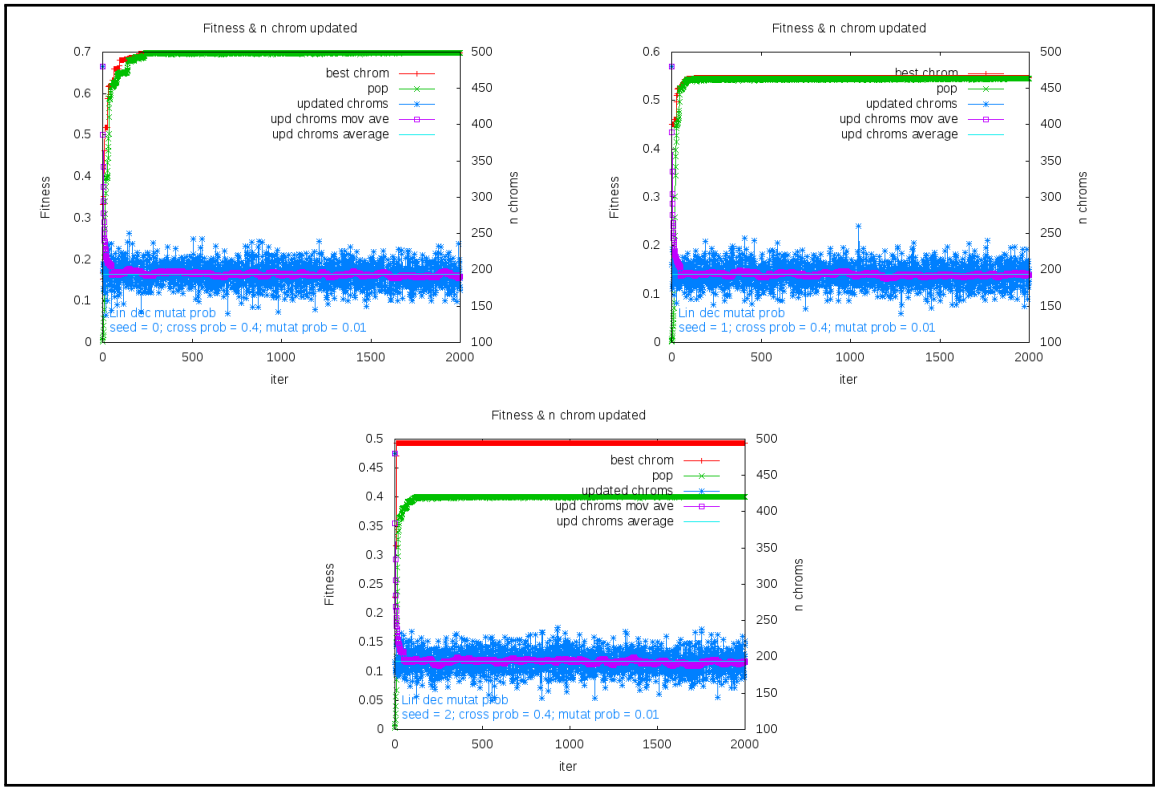


Figure A.49: Init mutation probability 0.01

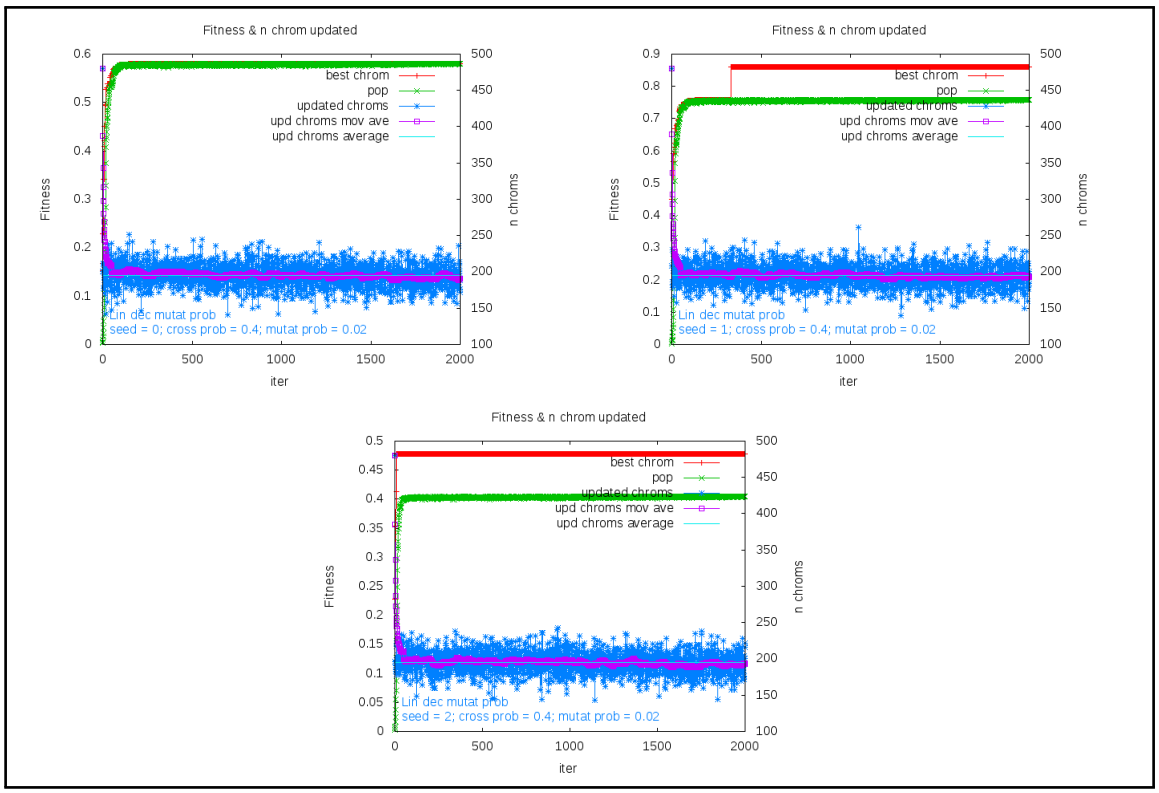


Figure A.50: Init mutation probability 0.02

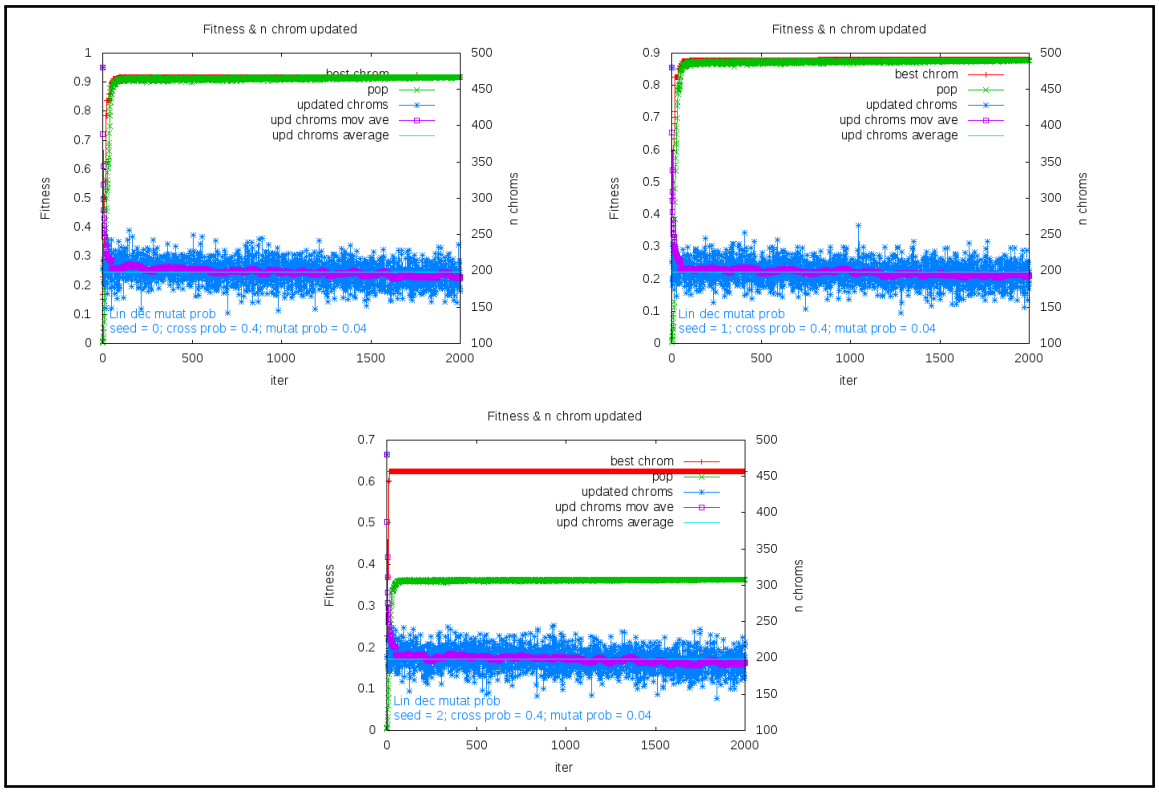


Figure A.51: Init mutation probability 0.04

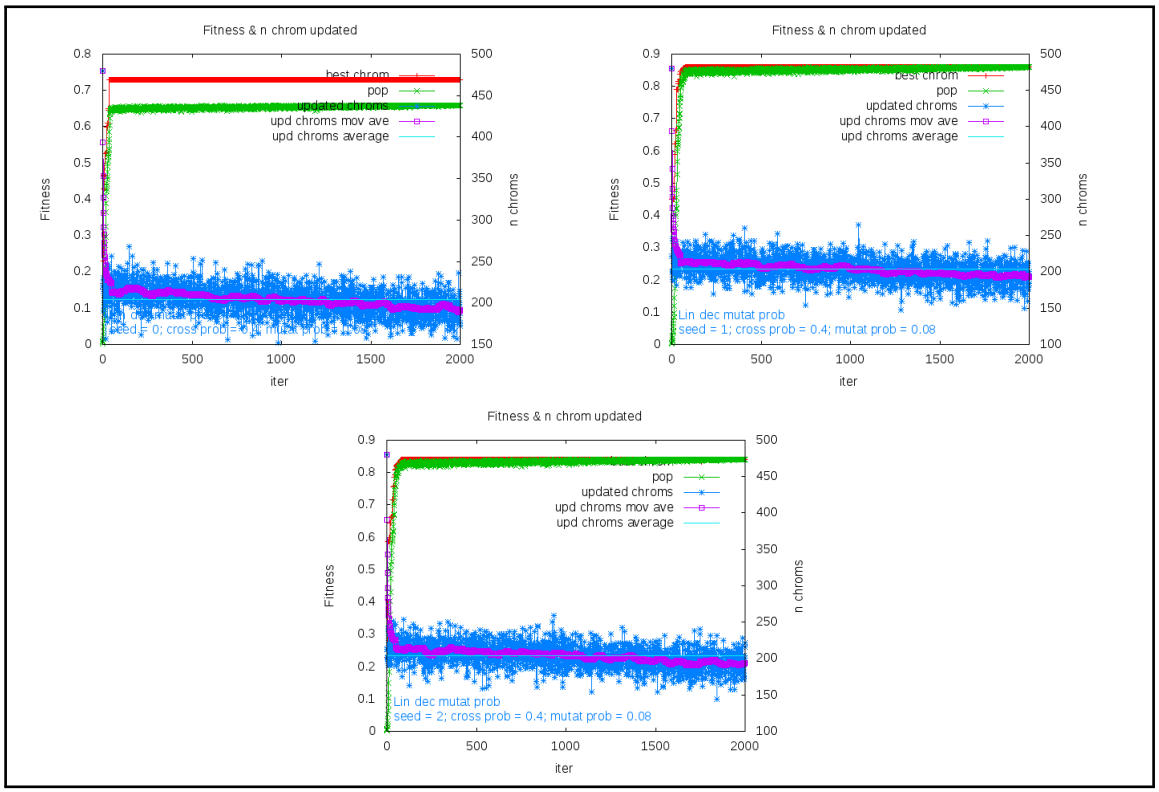


Figure A.52: Init mutation probability 0.08

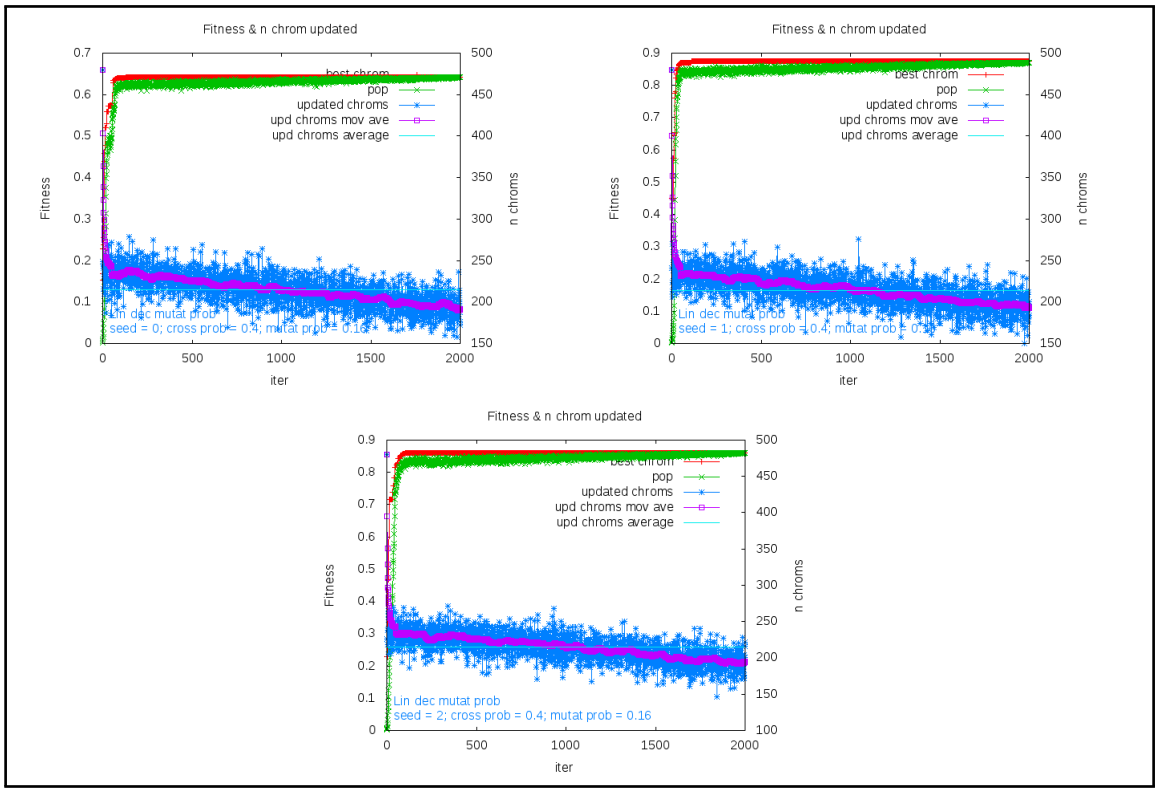


Figure A.53: Init mutation probability 0.16

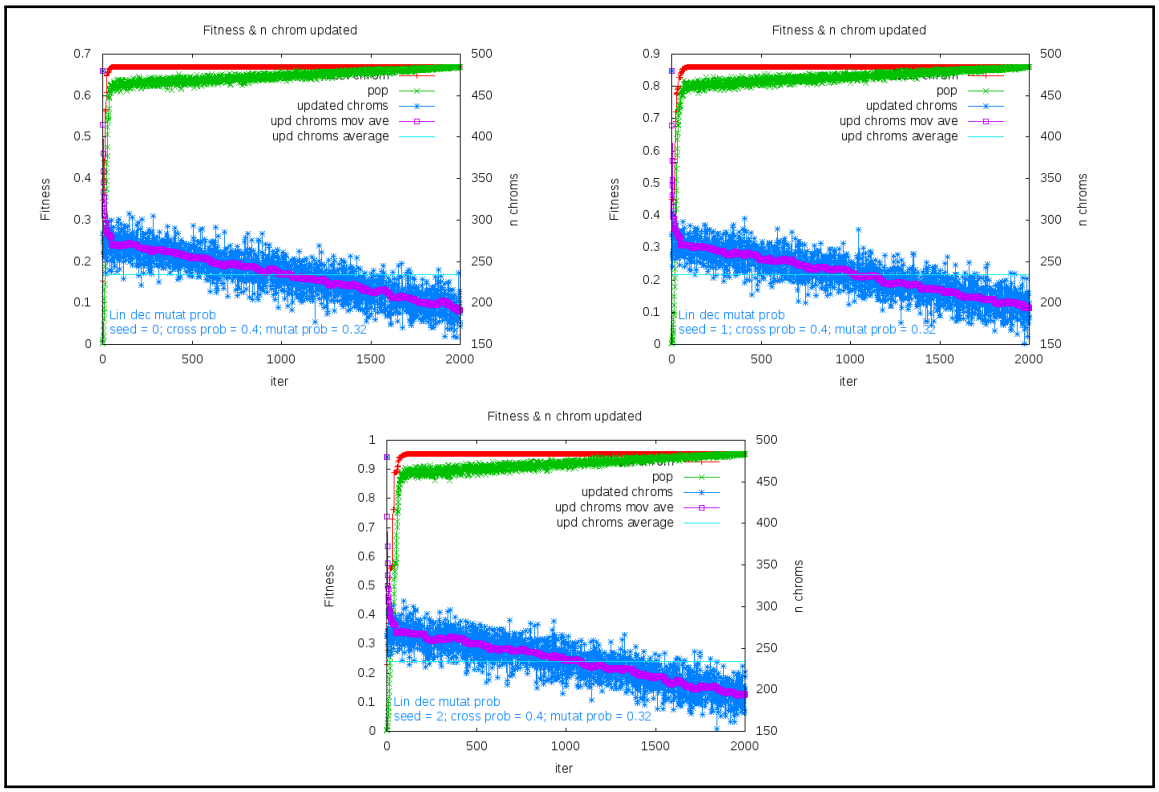


Figure A.54: Init mutation probability 0.32

A.2.3 Crossover probability 0.5

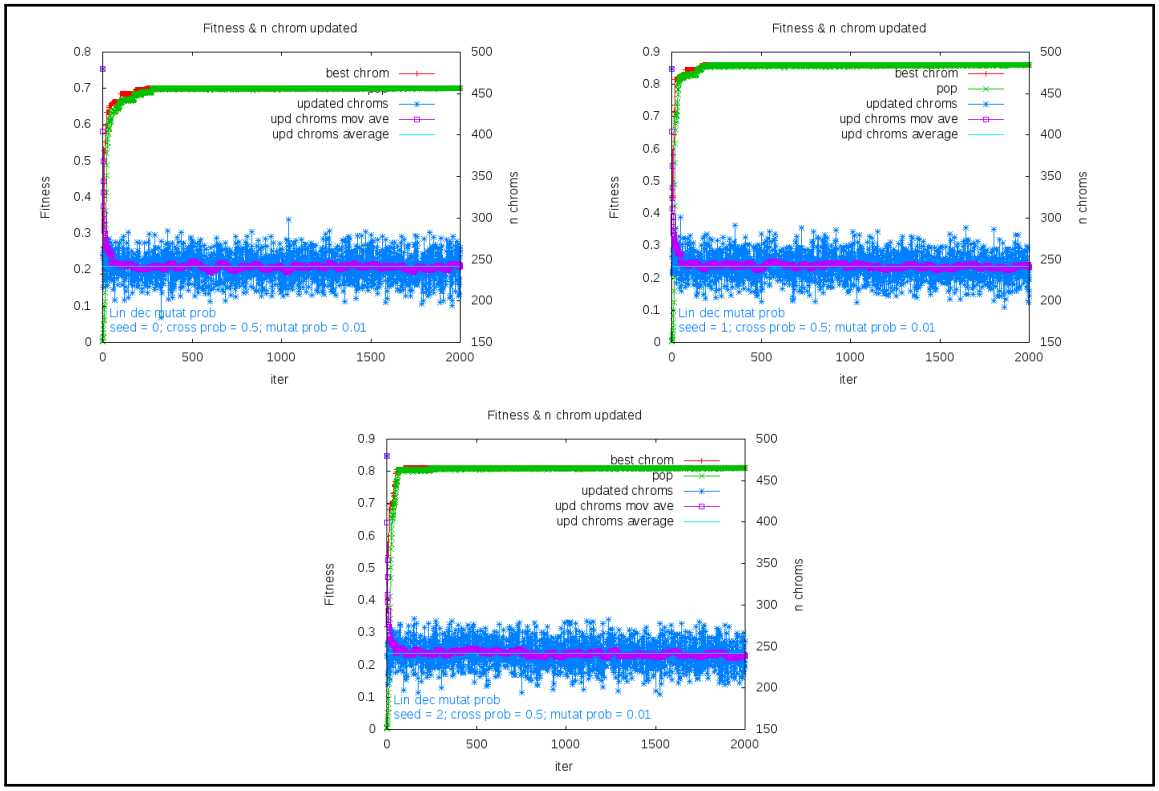


Figure A.55: Init mutation probability 0.01

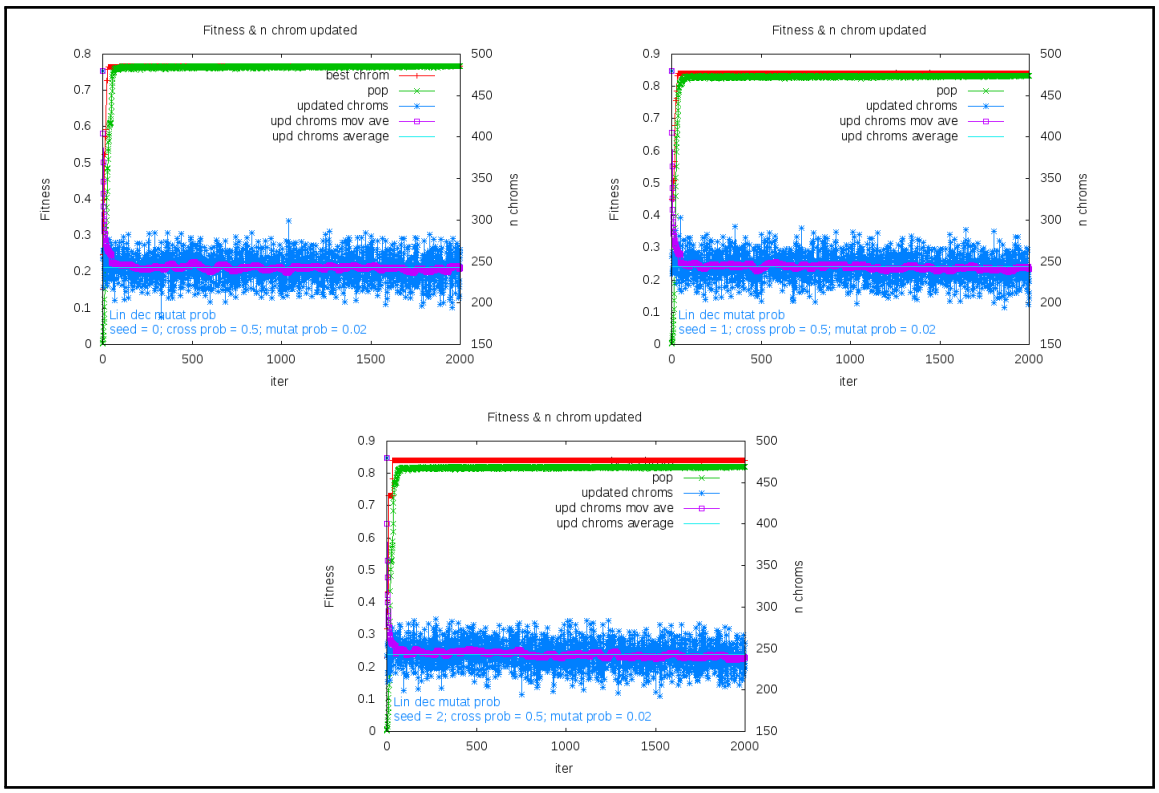


Figure A.56: Init mutation probability 0.02

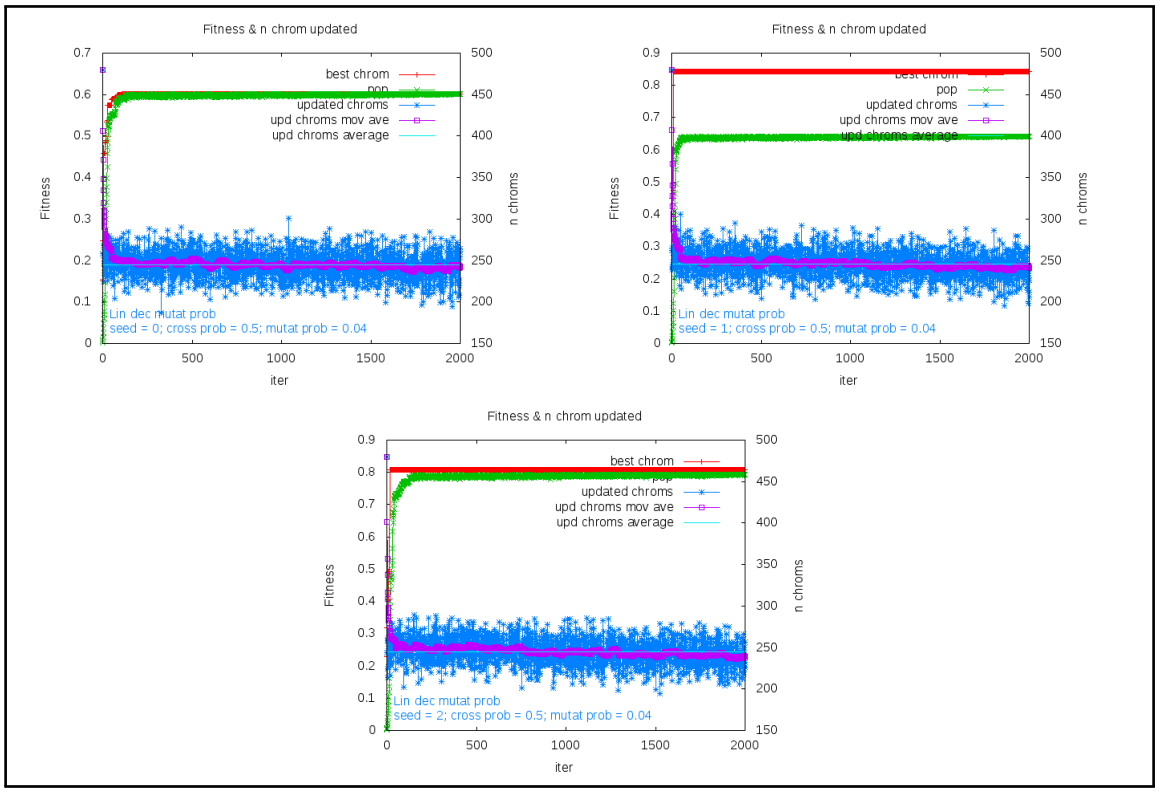


Figure A.57: Init mutation probability 0.04

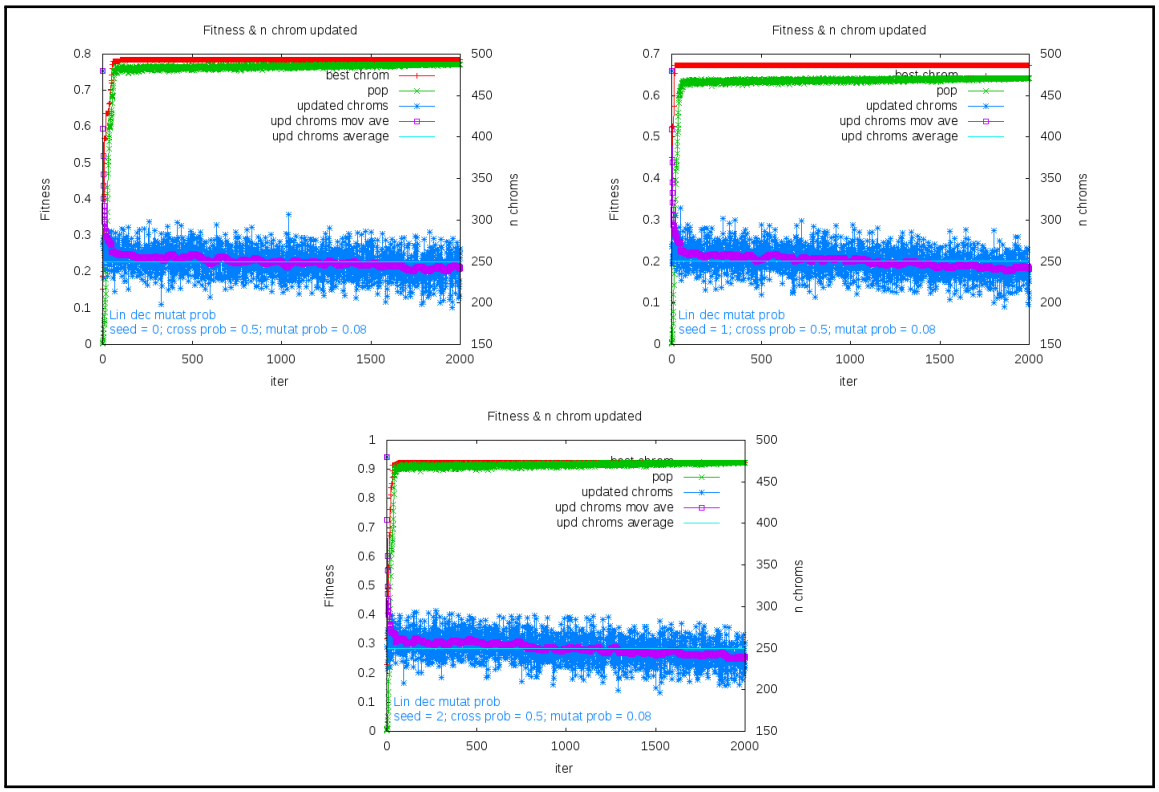


Figure A.58: Init mutation probability 0.08

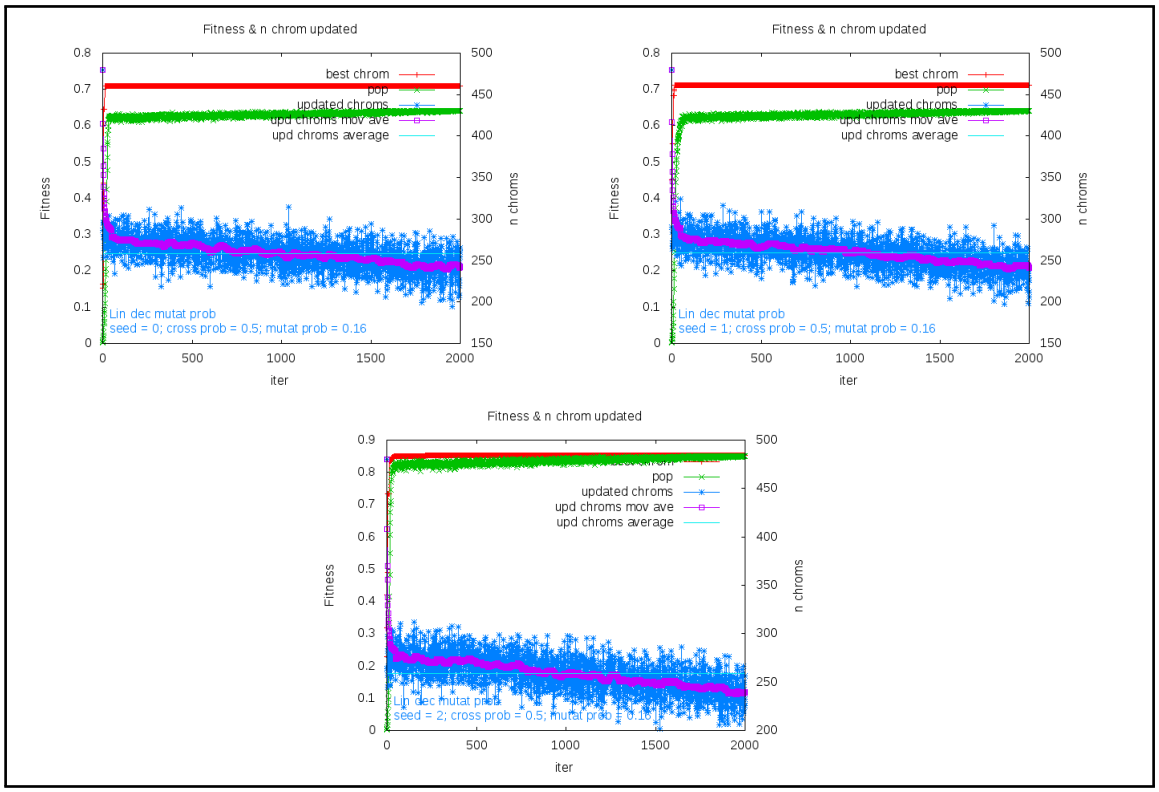


Figure A.59: Init mutation probability 0.16

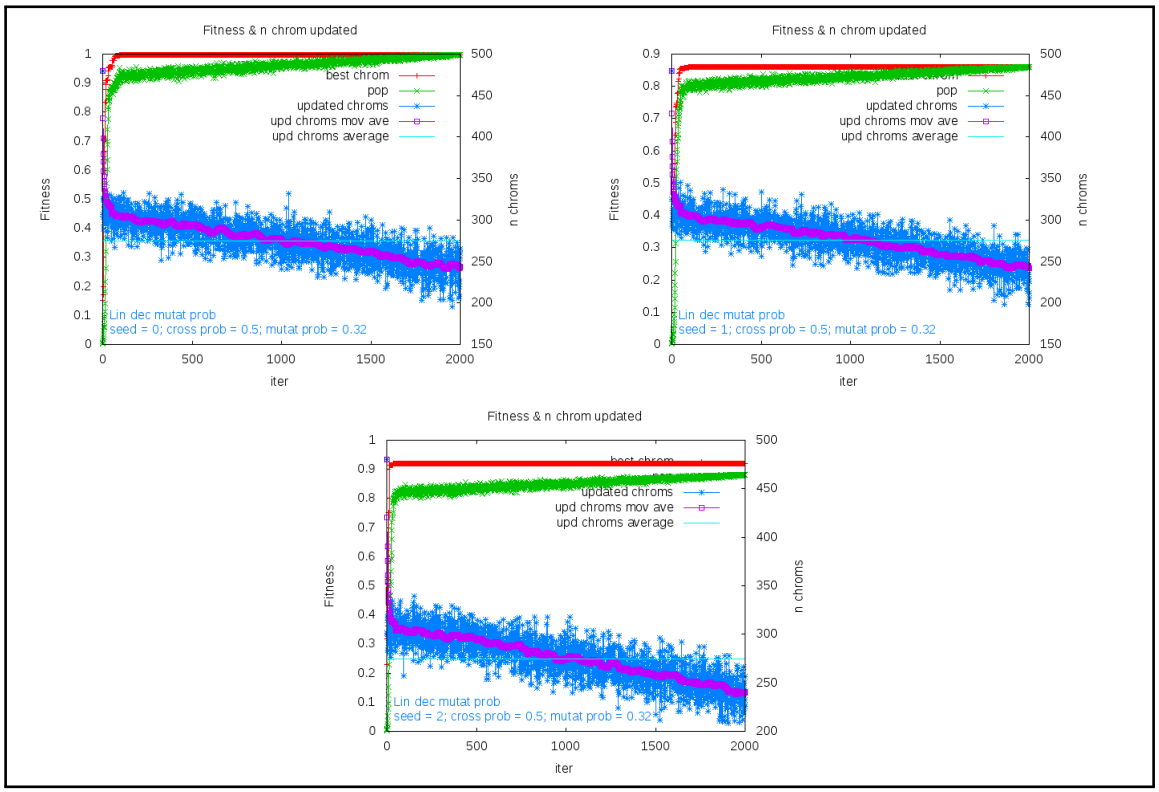


Figure A.60: Init mutation probability 0.32

A.2.4 Crossover probability 0.6

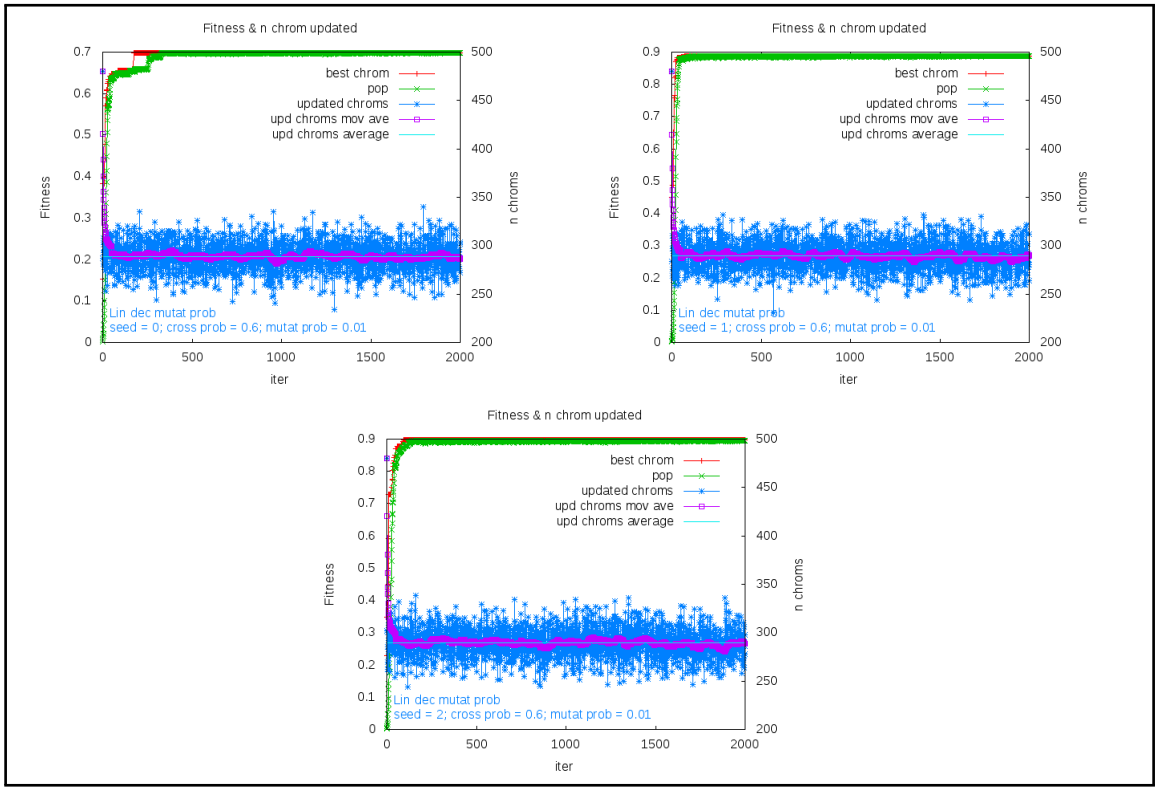


Figure A.61: Init mutation probability 0.01

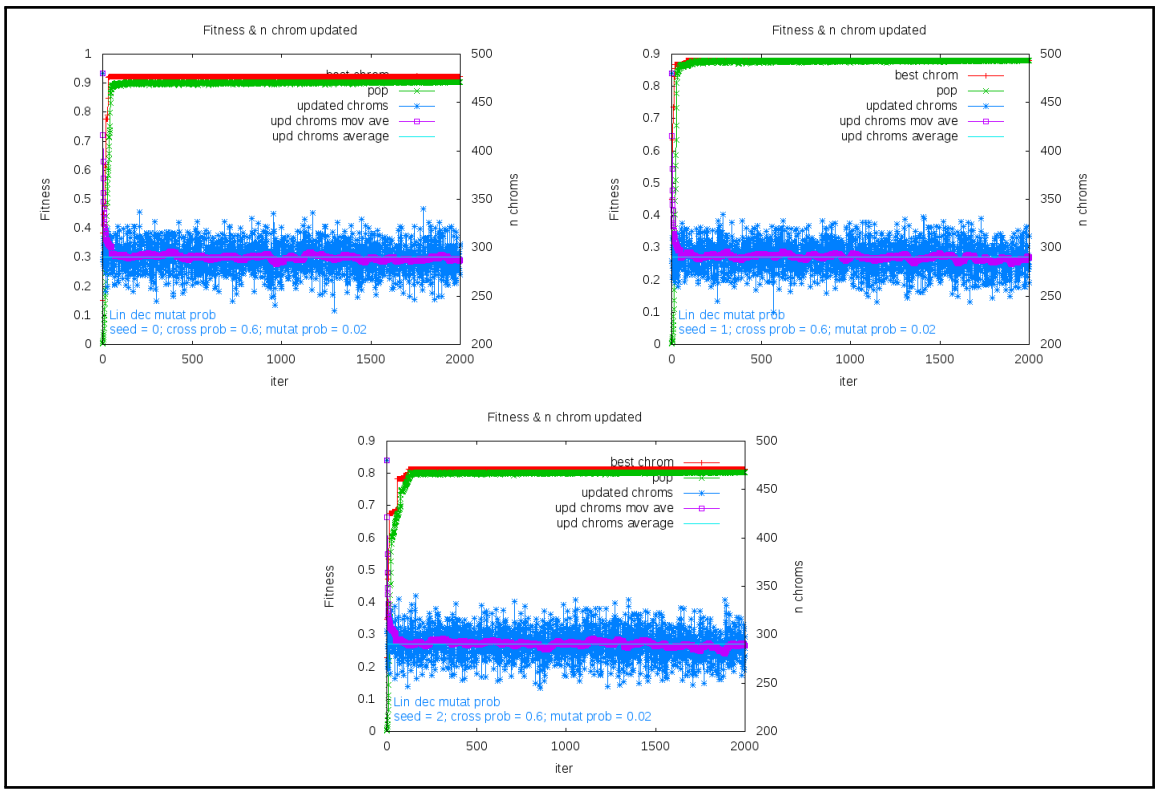


Figure A.62: Init mutation probability 0.02

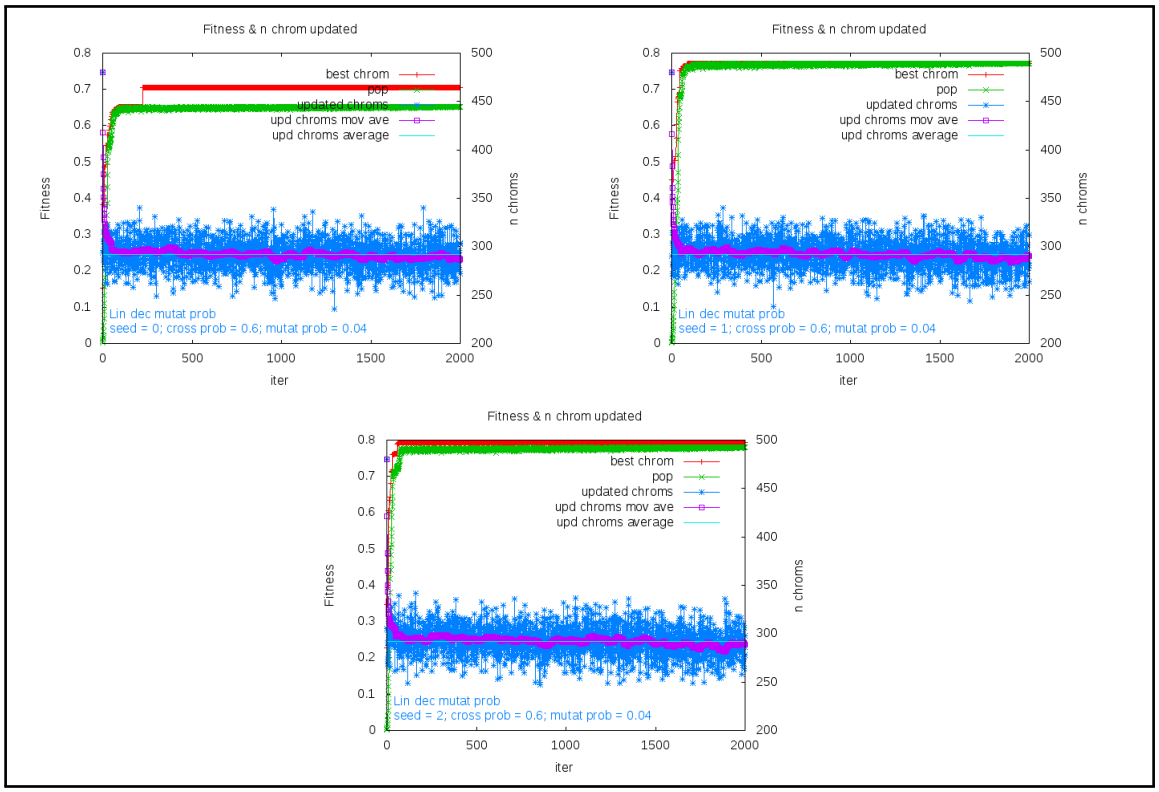


Figure A.63: Init mutation probability 0.04

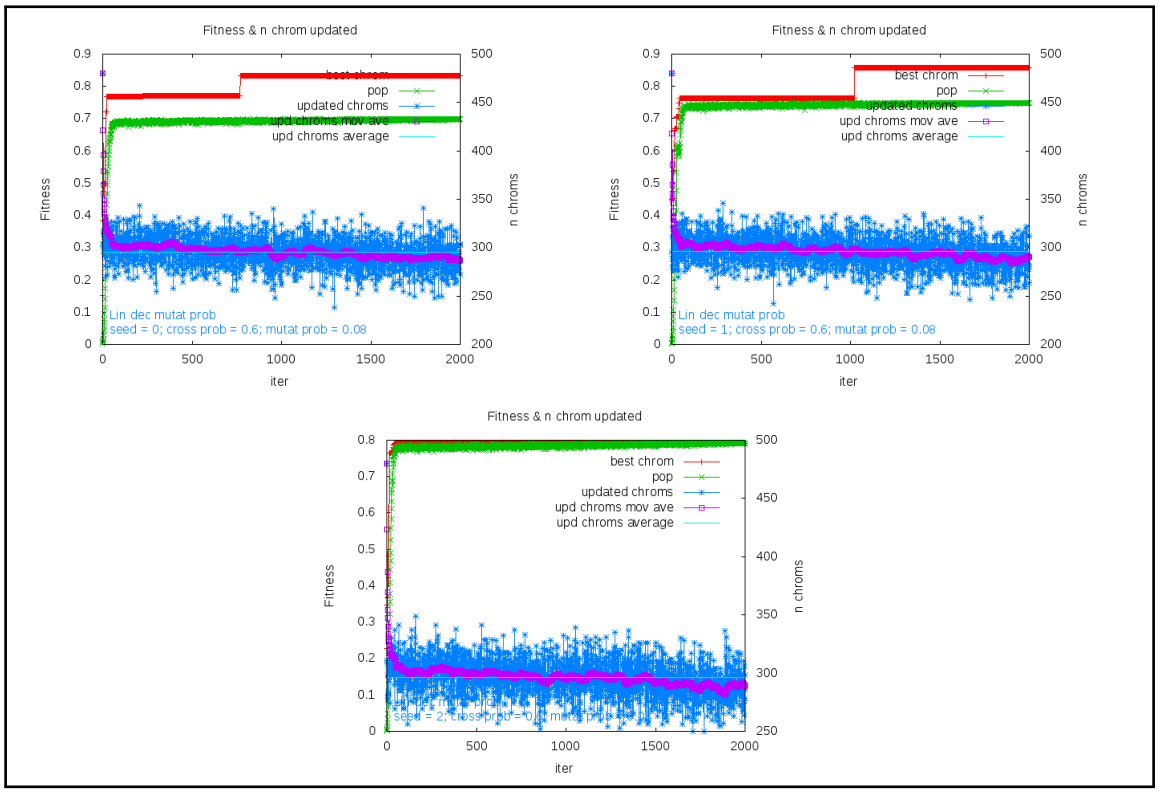


Figure A.64: Init mutation probability 0.08

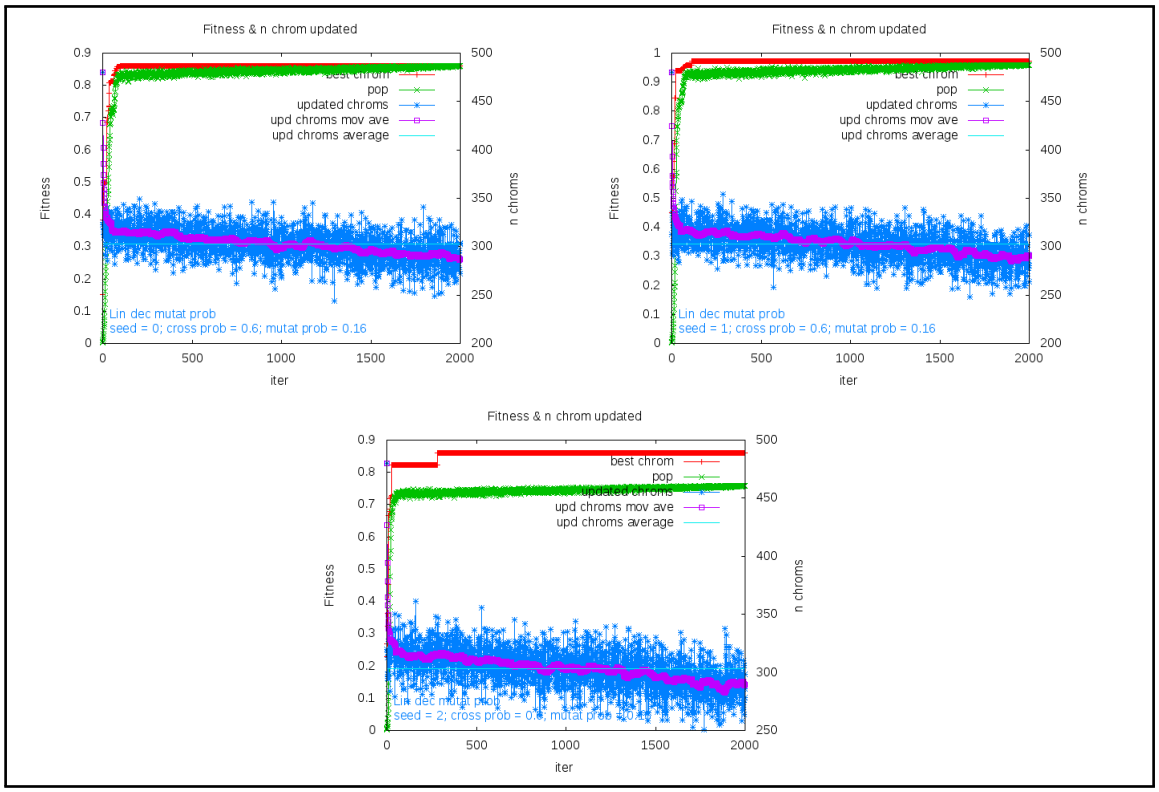


Figure A.65: Init mutation probability 0.16

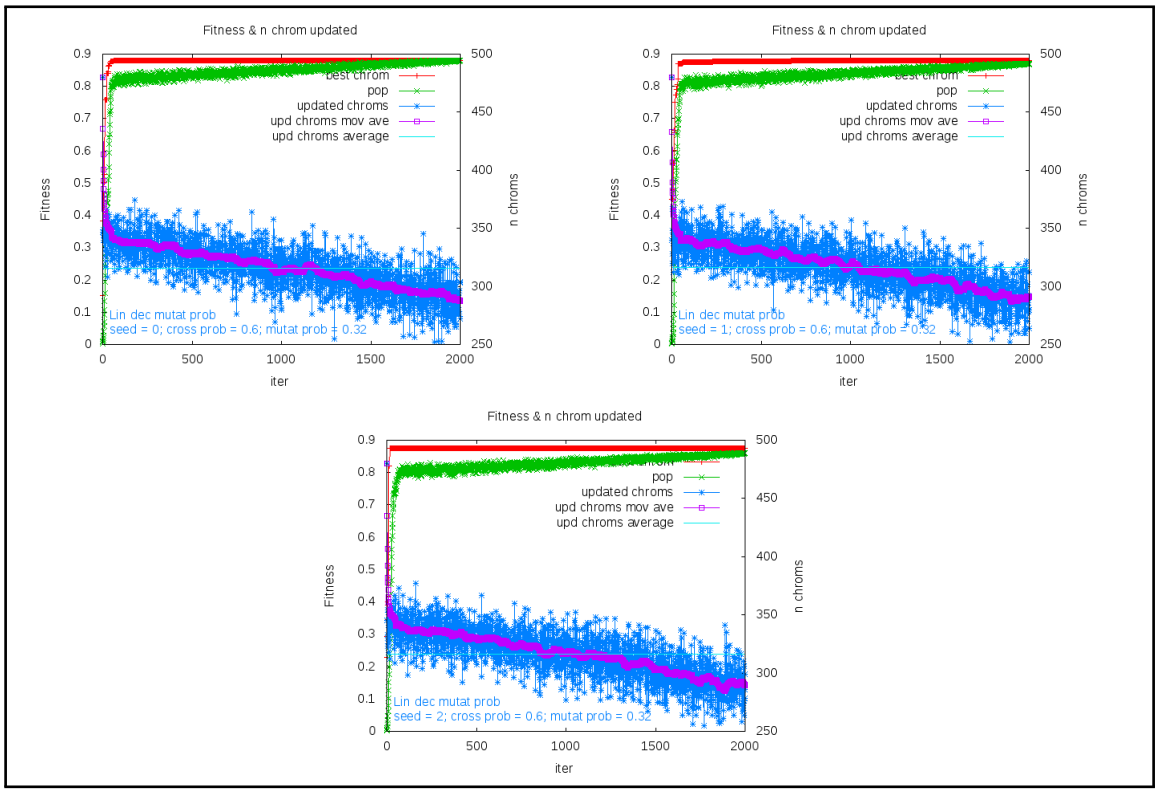


Figure A.66: Init mutation probability 0.32

A.2.5 Crossover probability 0.7

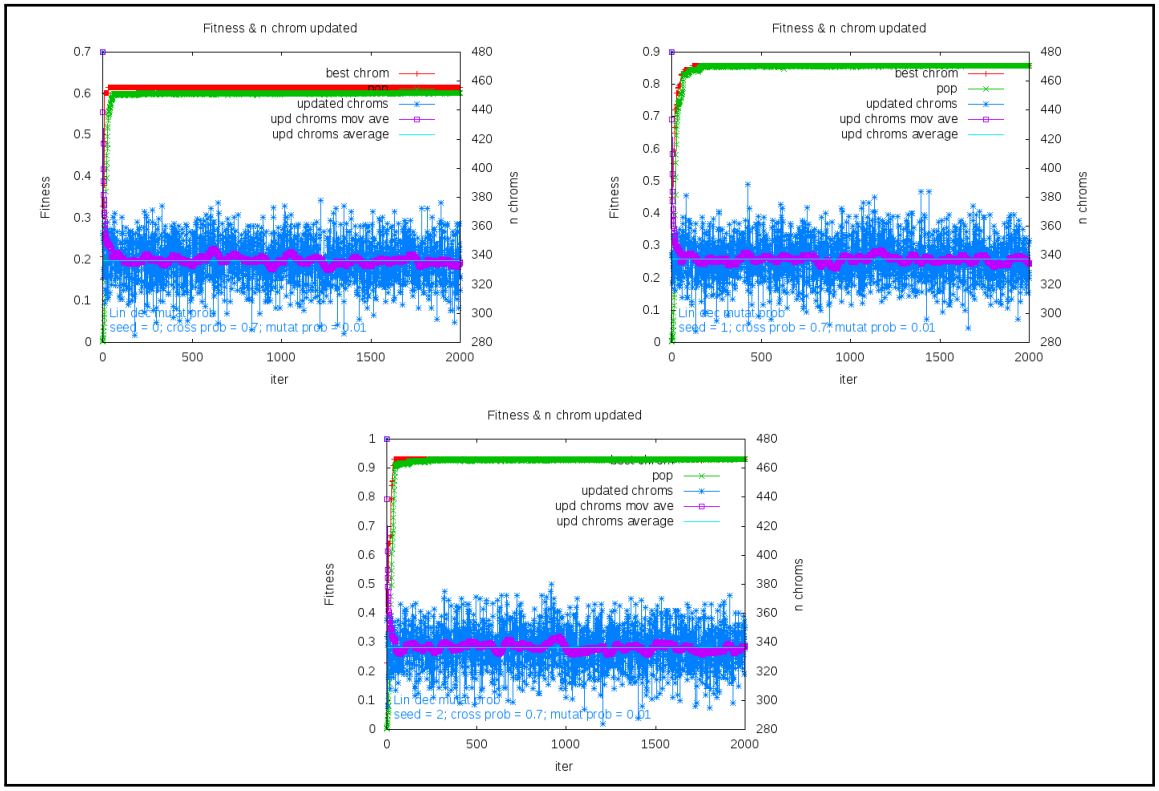


Figure A.67: Init mutation probability 0.01

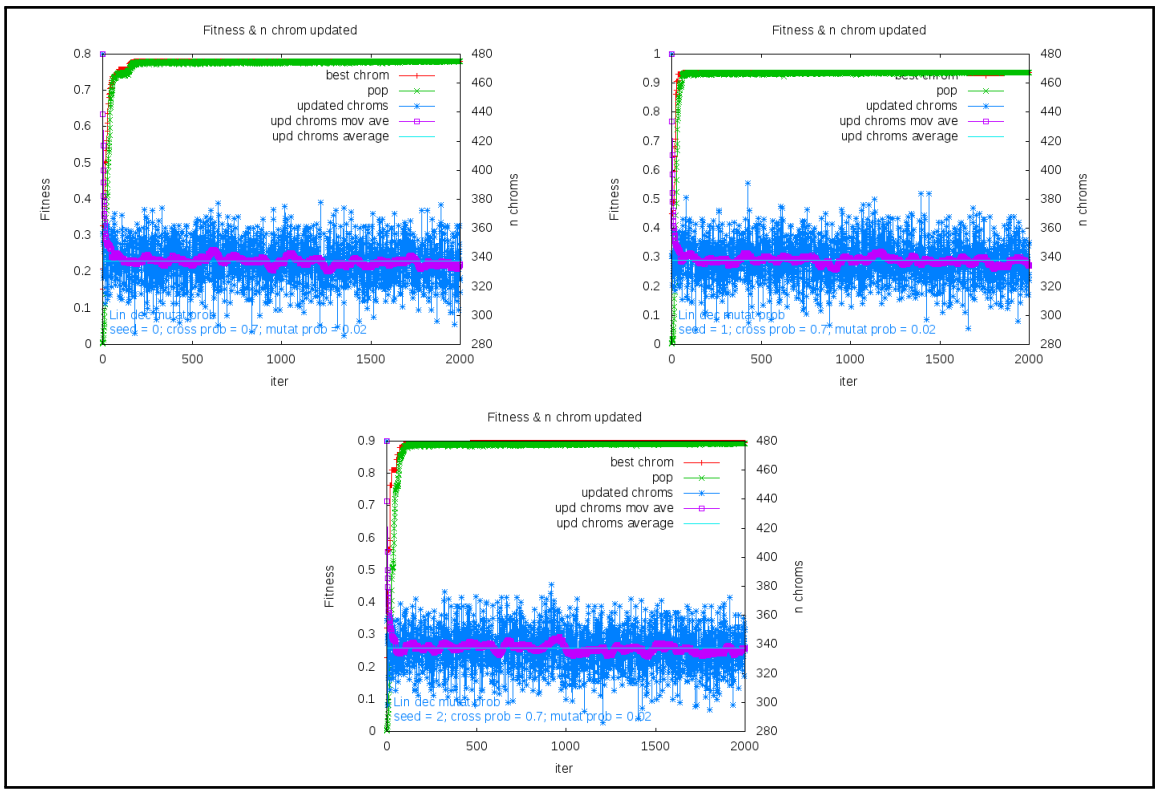


Figure A.68: Init mutation probability 0.02

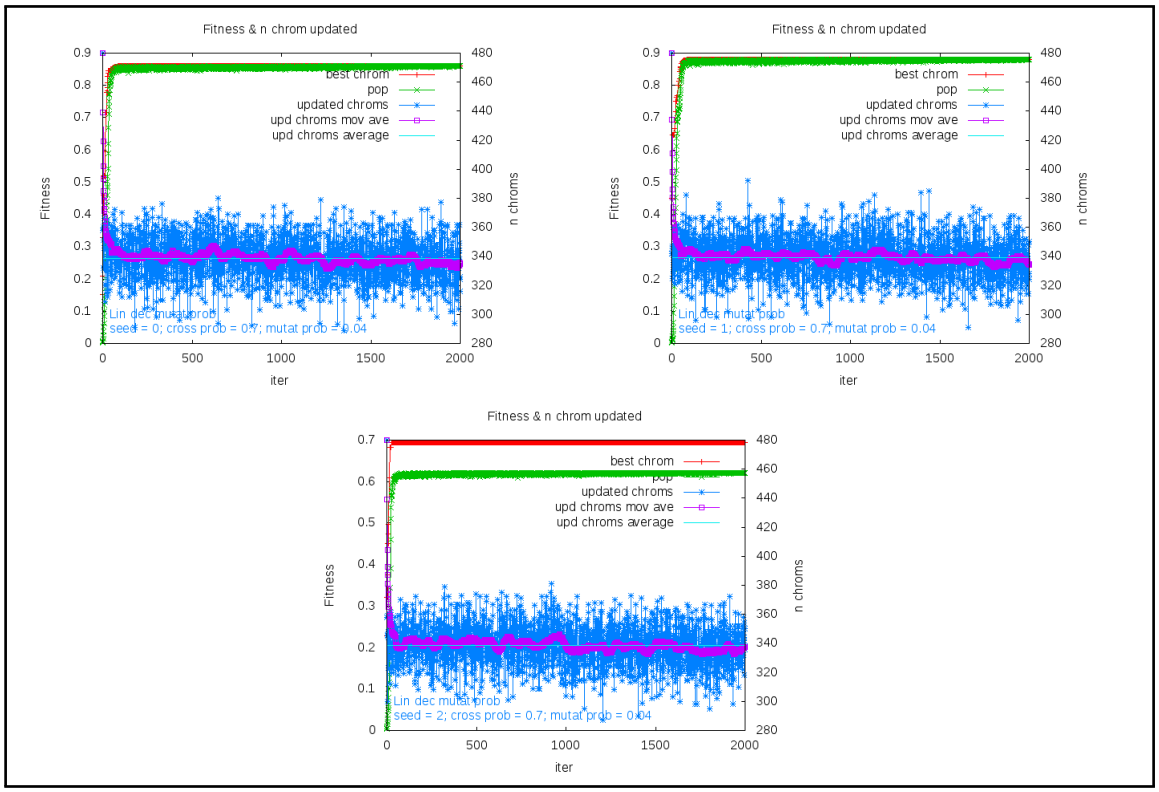


Figure A.69: Init mutation probability 0.04

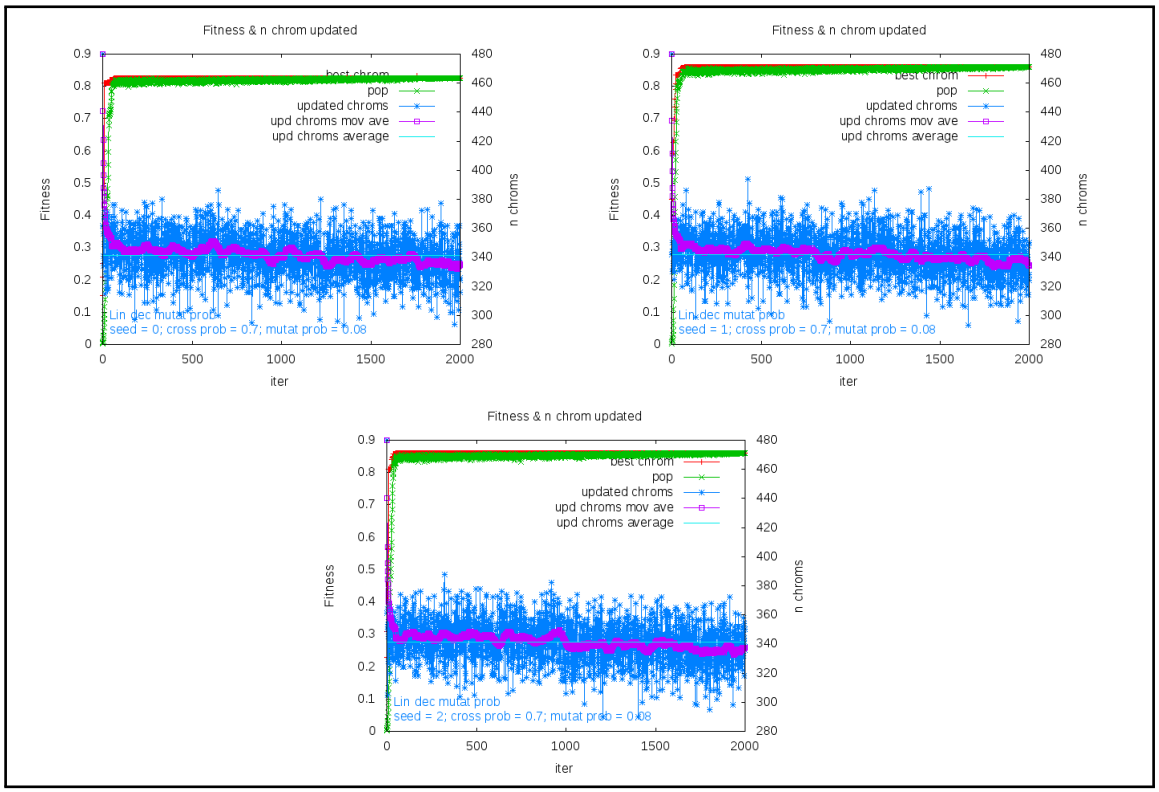


Figure A.70: Init mutation probability 0.08

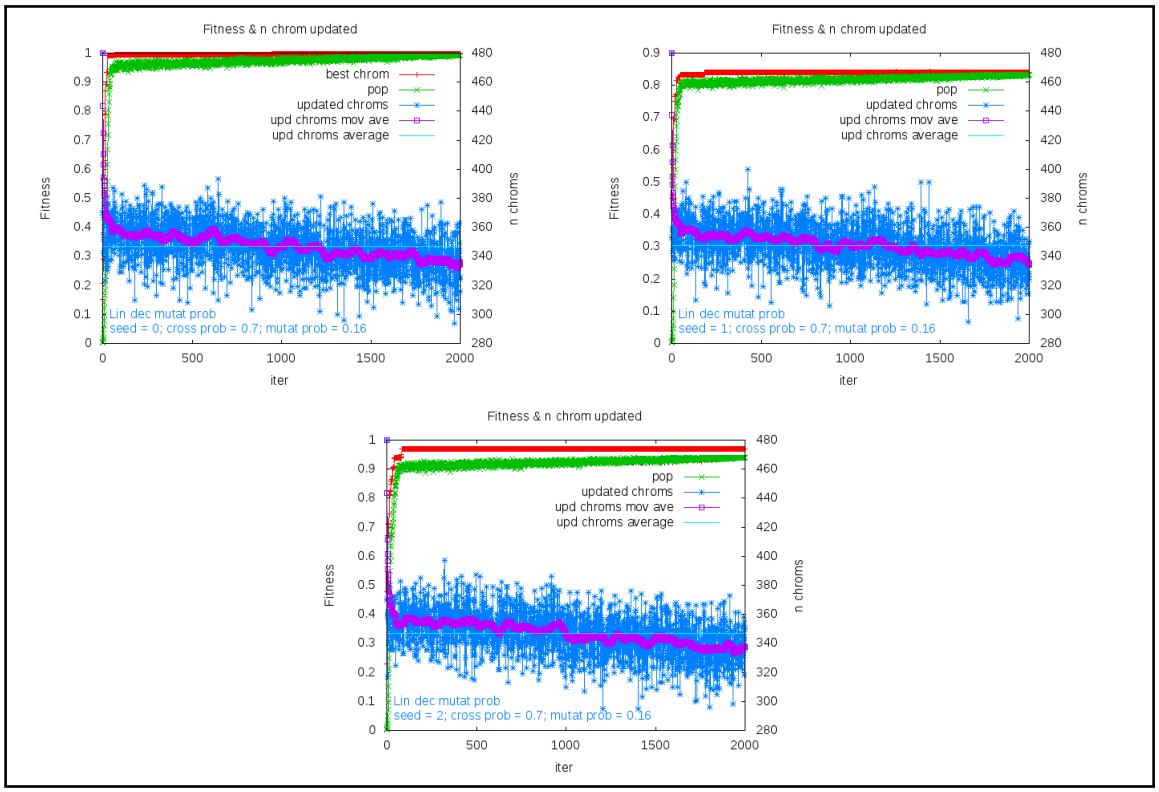


Figure A.71: Init mutation probability 0.16

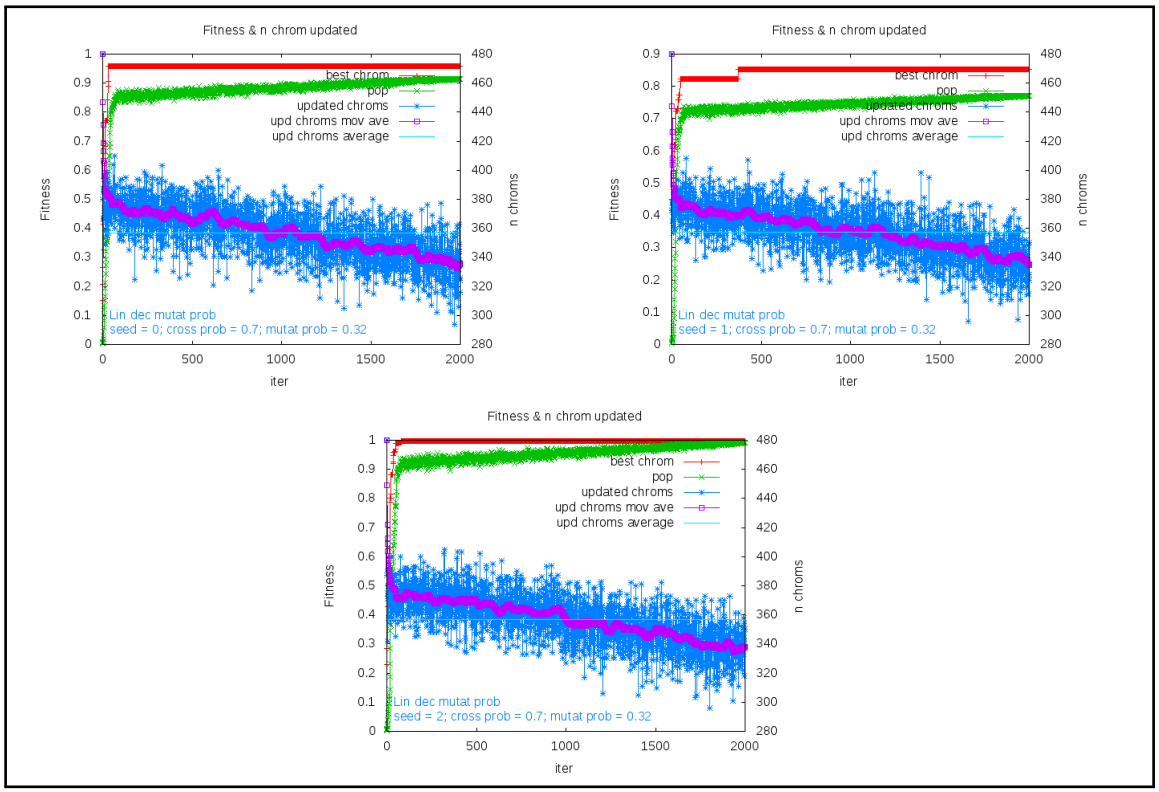


Figure A.72: Init mutation probability 0.32

A.2.6 Crossover probability 0.8

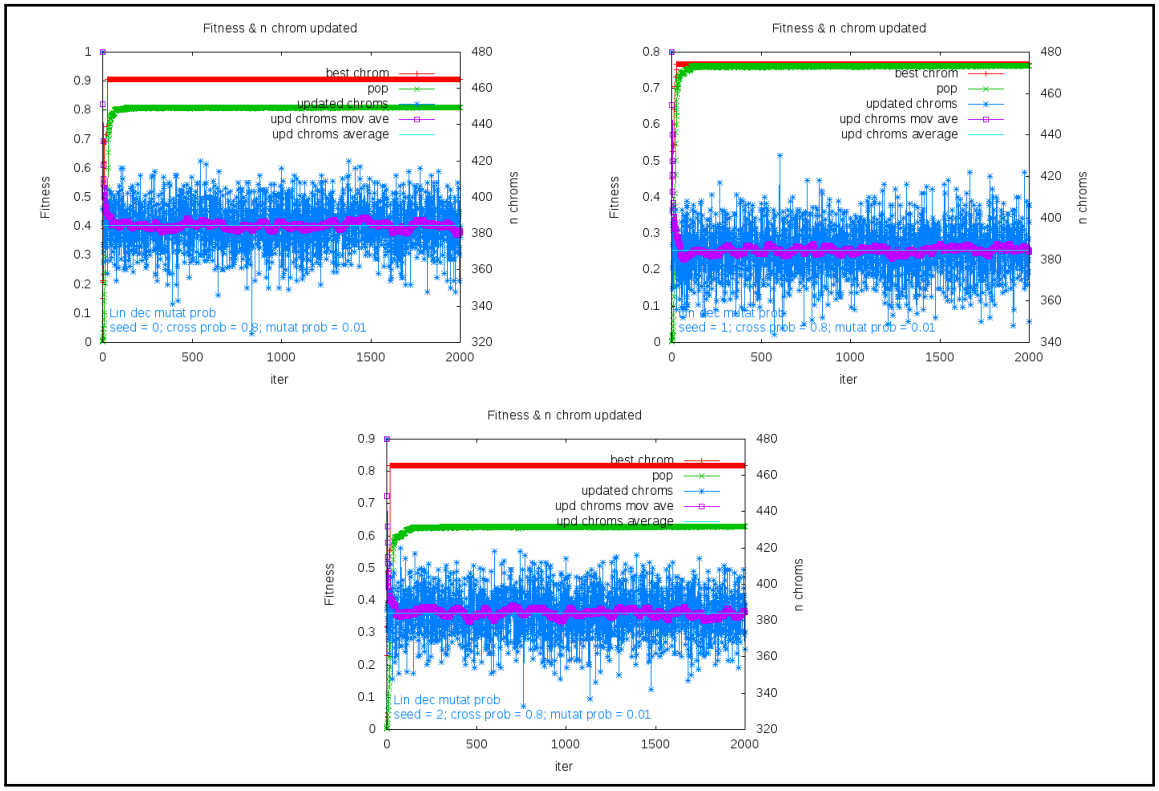


Figure A.73: Init mutation probability 0.01

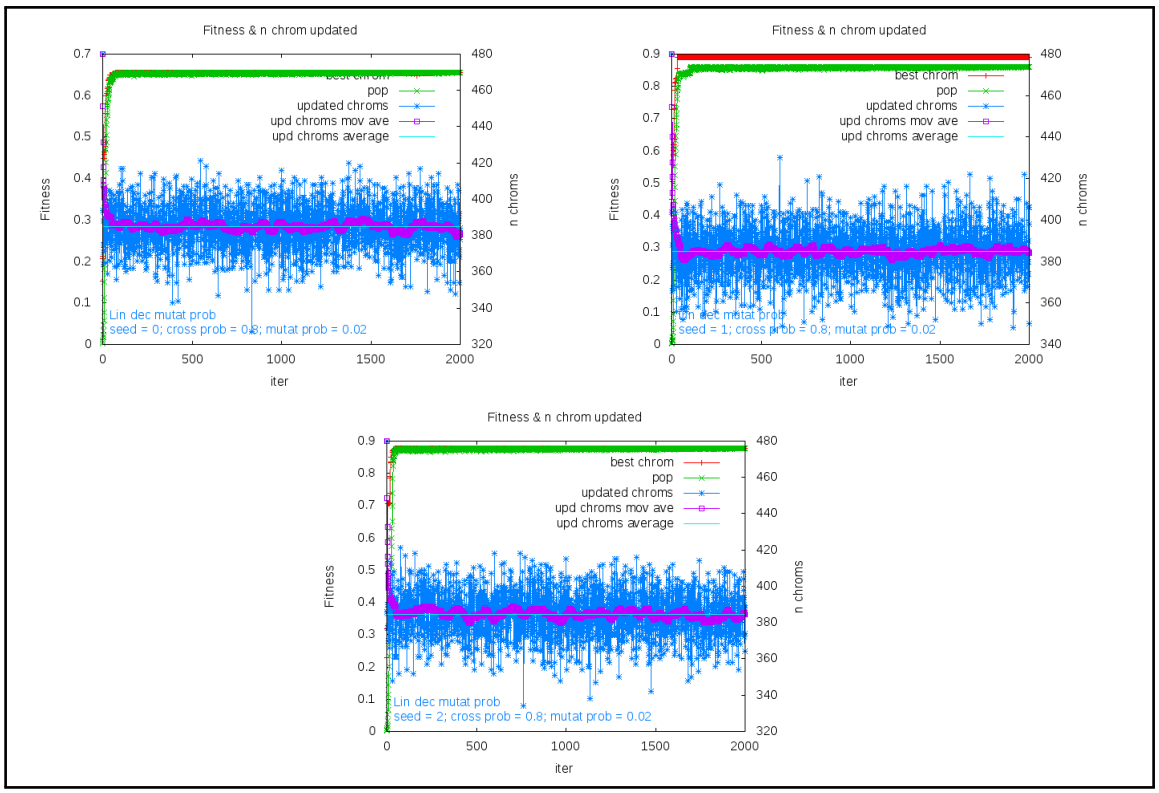


Figure A.74: Init mutation probability 0.02

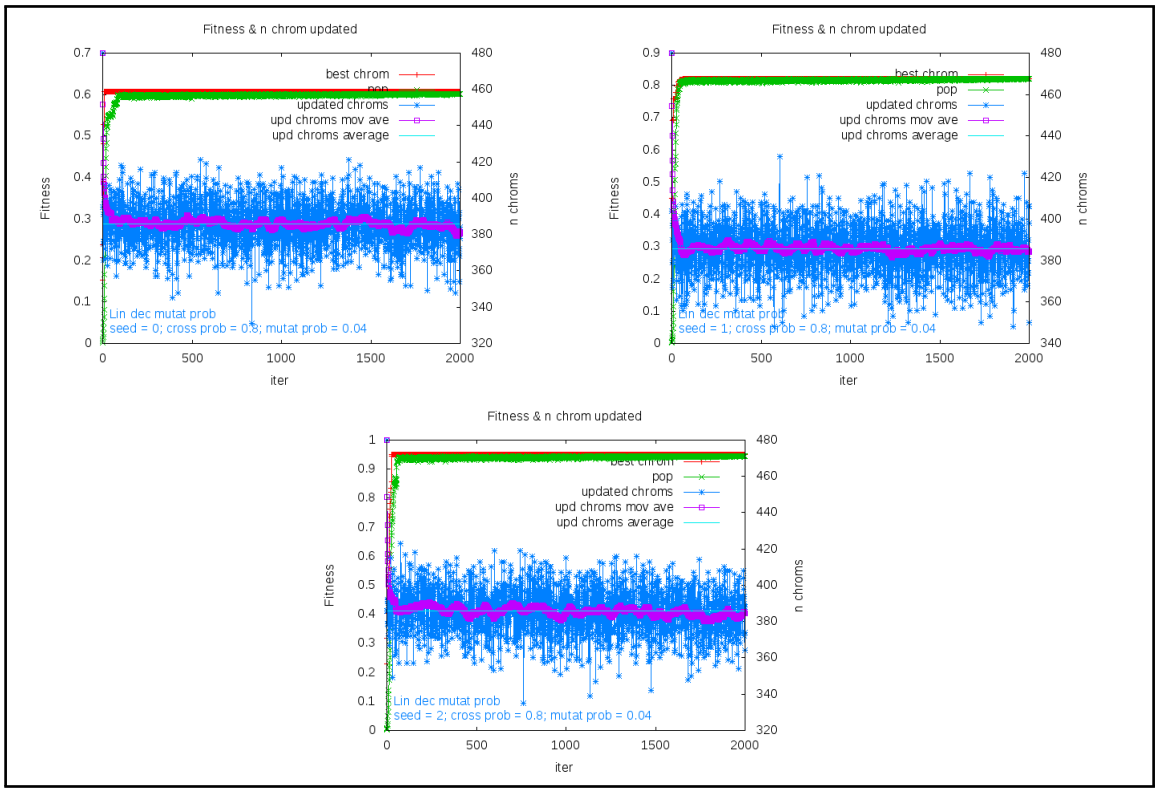


Figure A.75: Init mutation probability 0.04

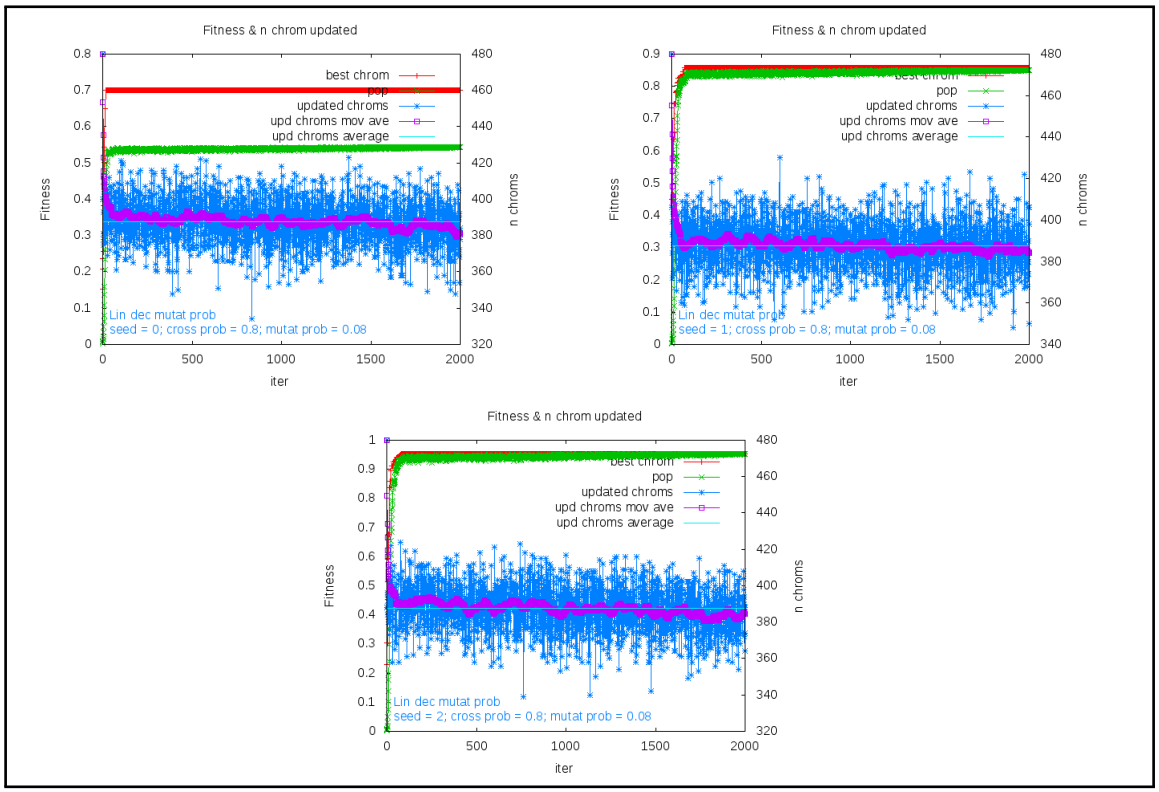


Figure A.76: Init mutation probability 0.08

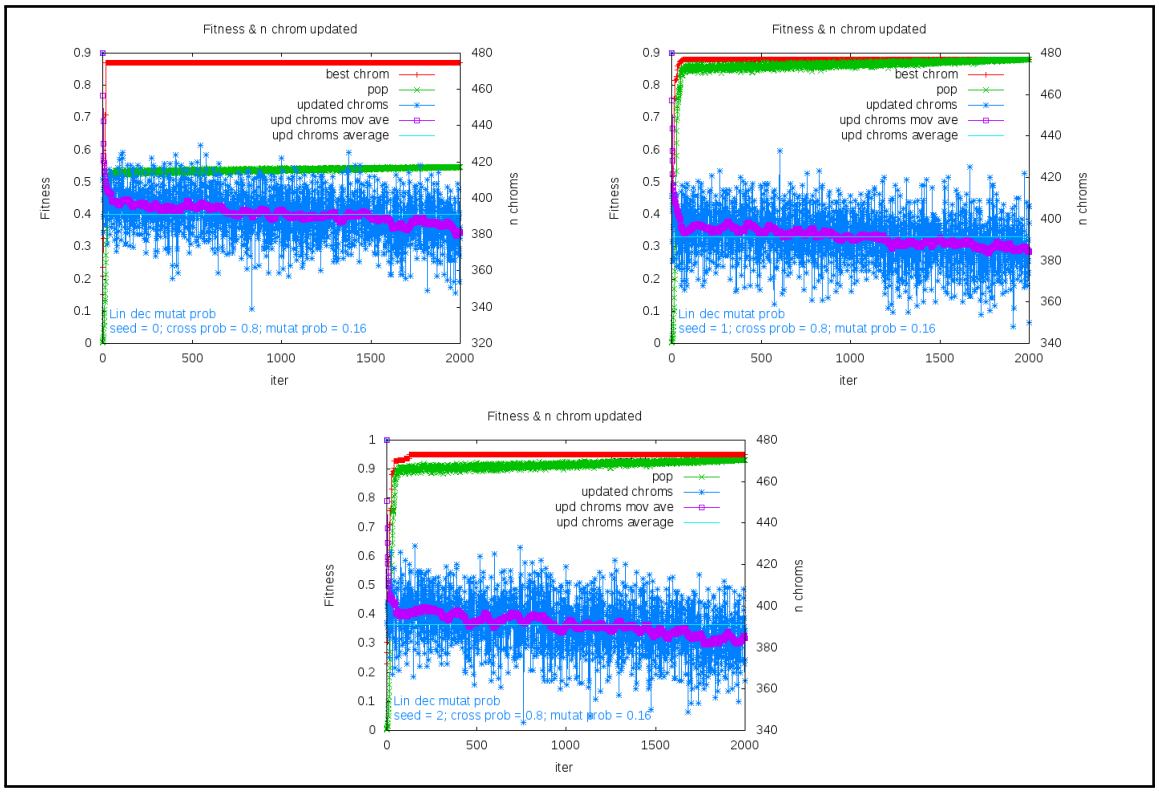


Figure A.77: Init mutation probability 0.16

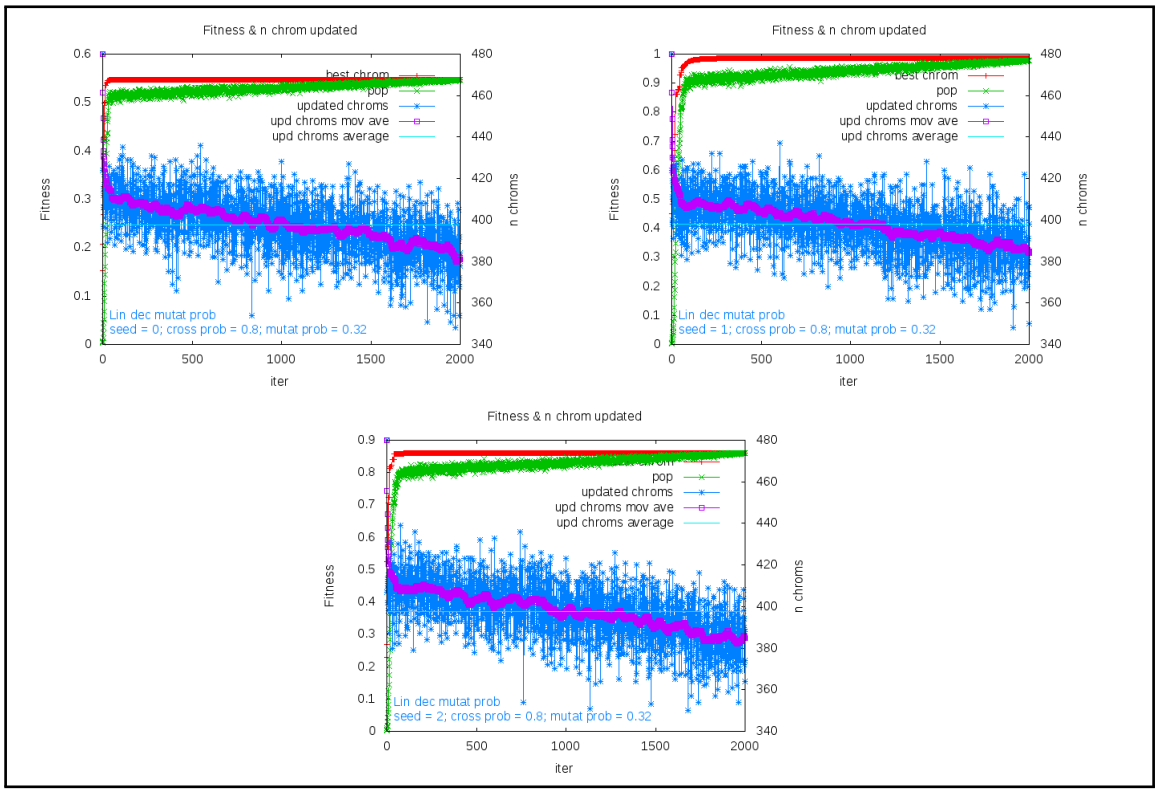


Figure A.78: Init mutation probability 0.32

A.2.7 Crossover probability 0.9

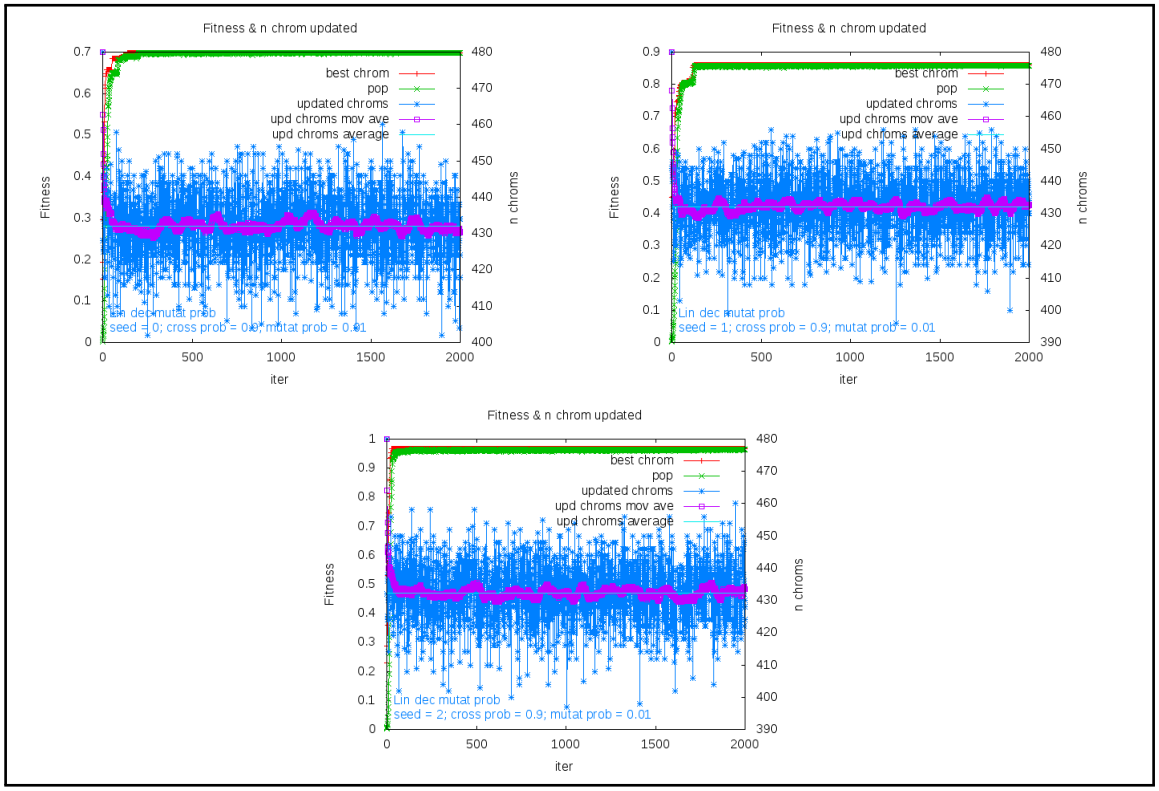


Figure A.79: Init mutation probability 0.01

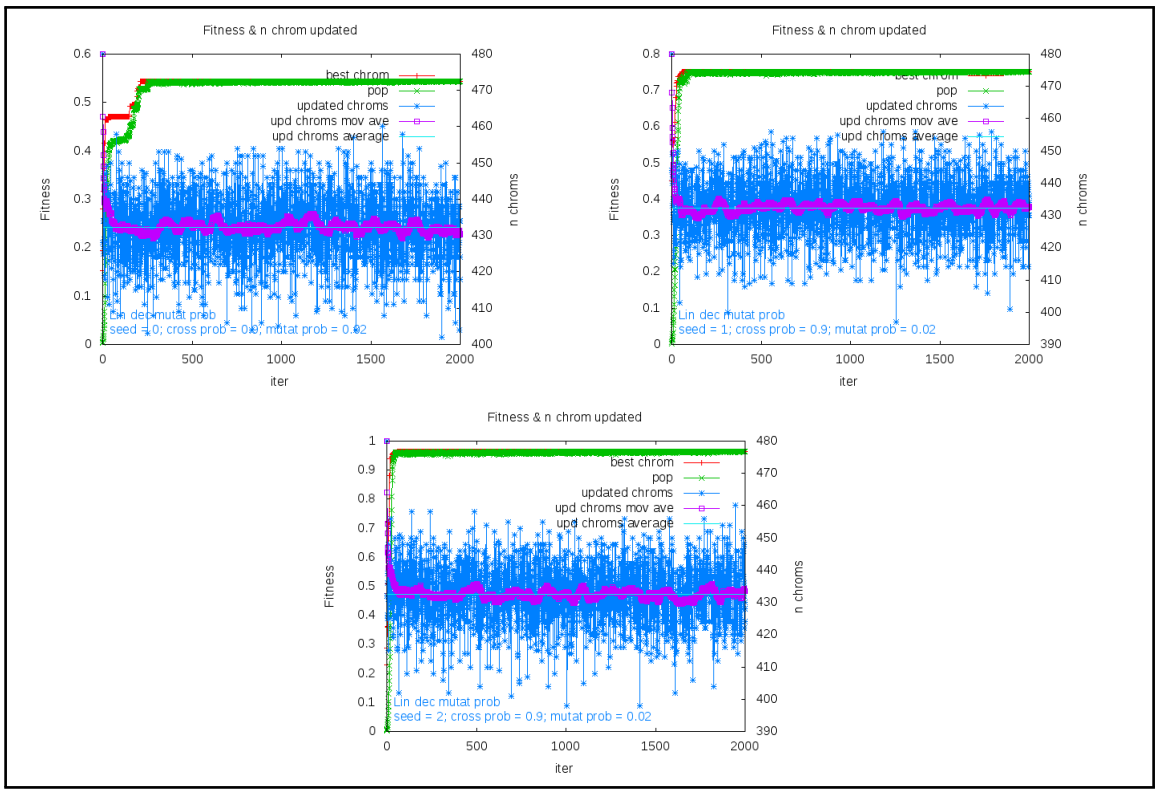


Figure A.80: Init mutation probability 0.02

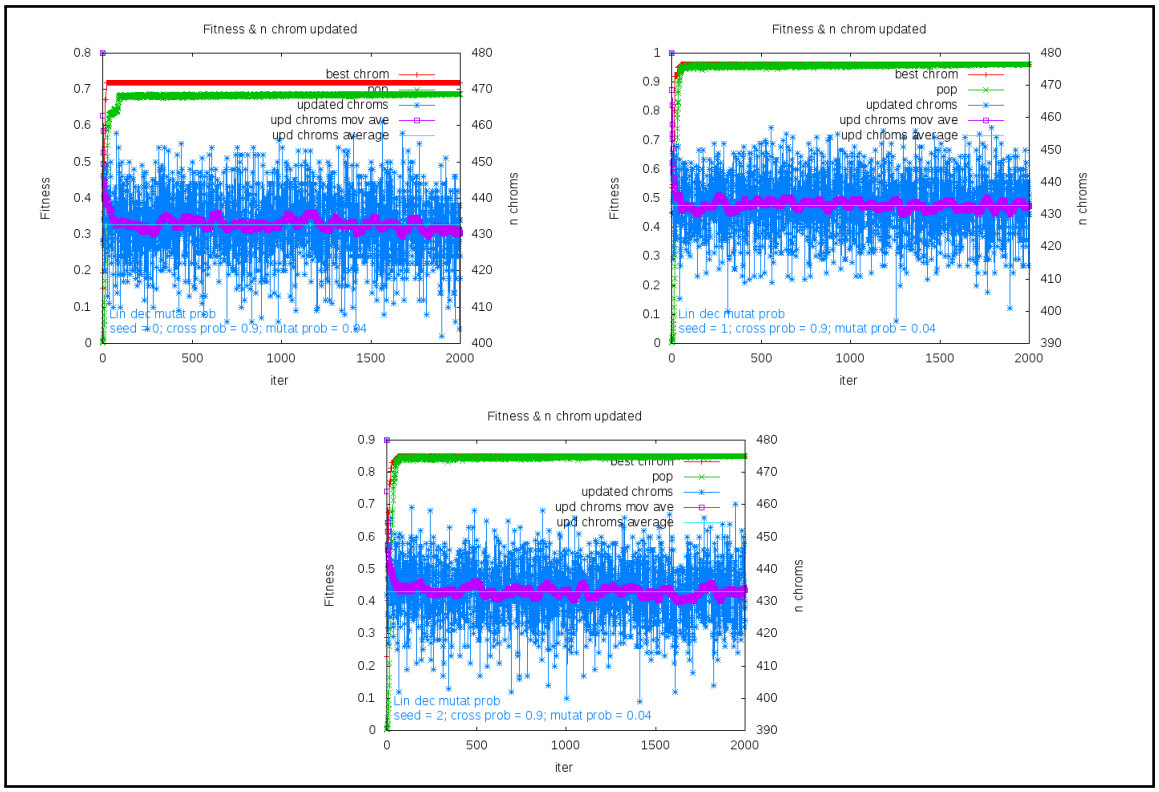


Figure A.81: Init mutation probability 0.04

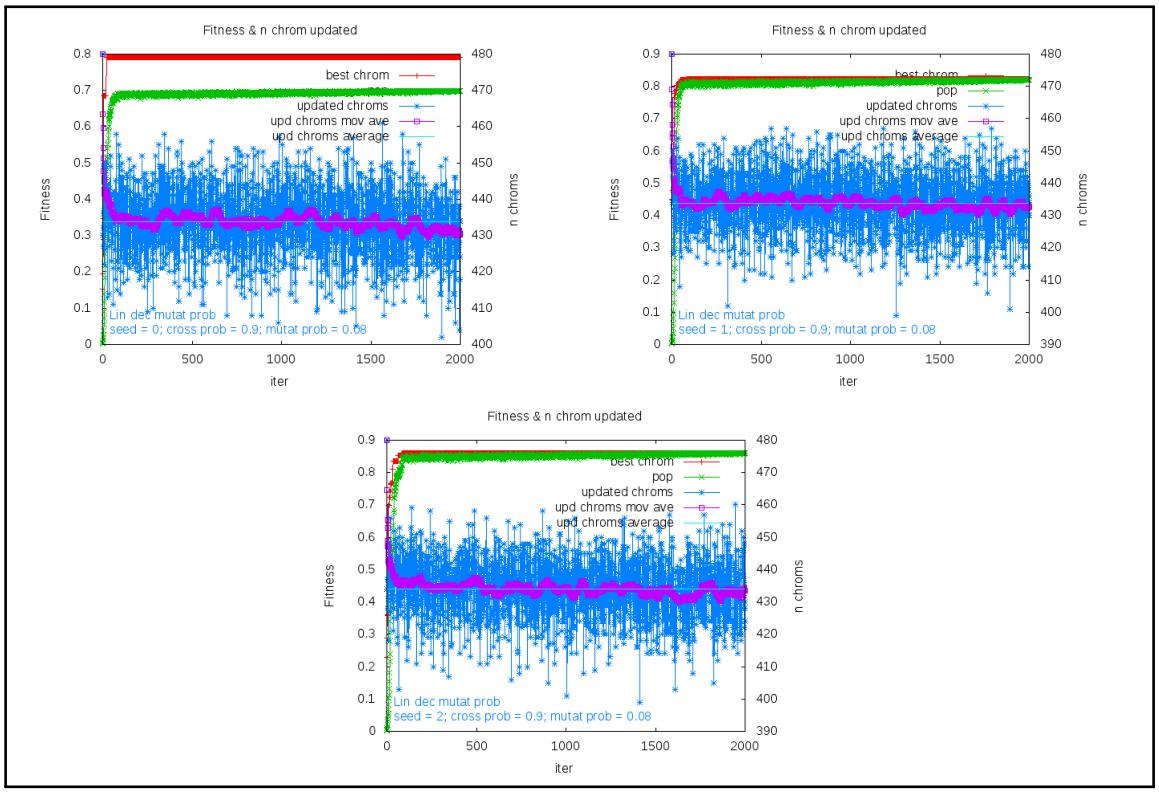


Figure A.82: Init mutation probability 0.08

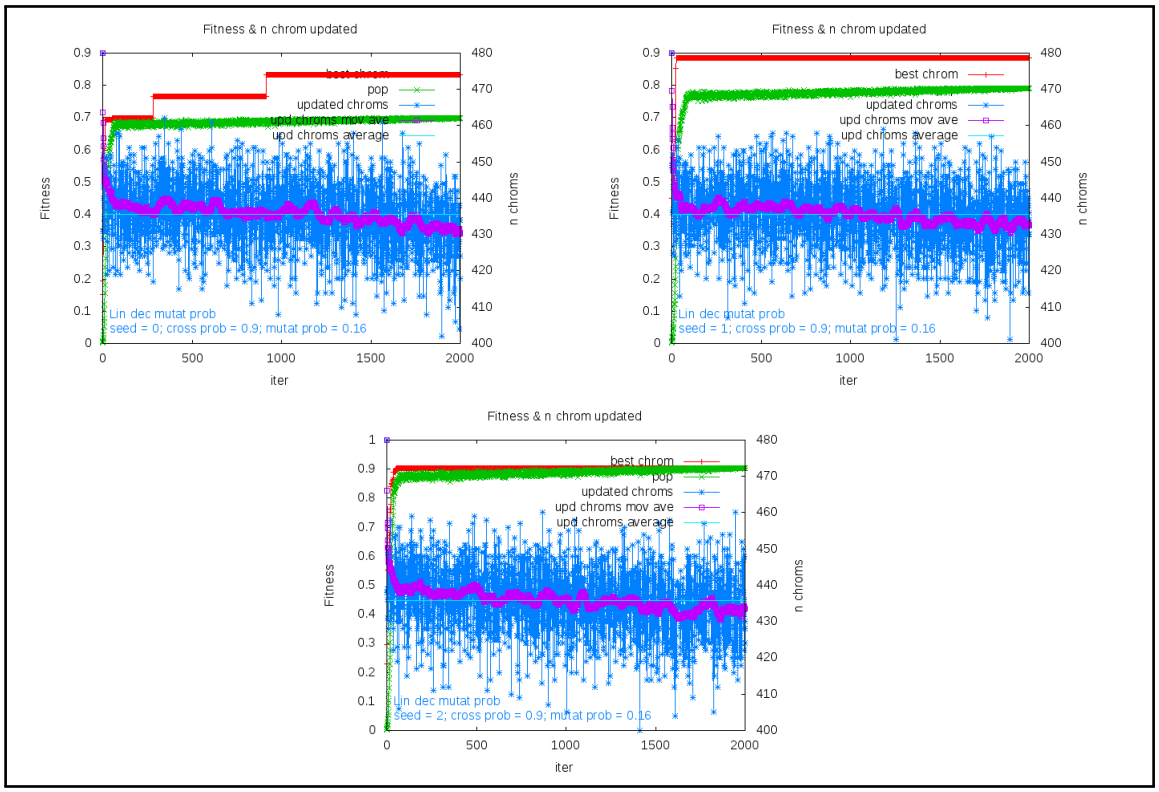


Figure A.83: Init mutation probability 0.16

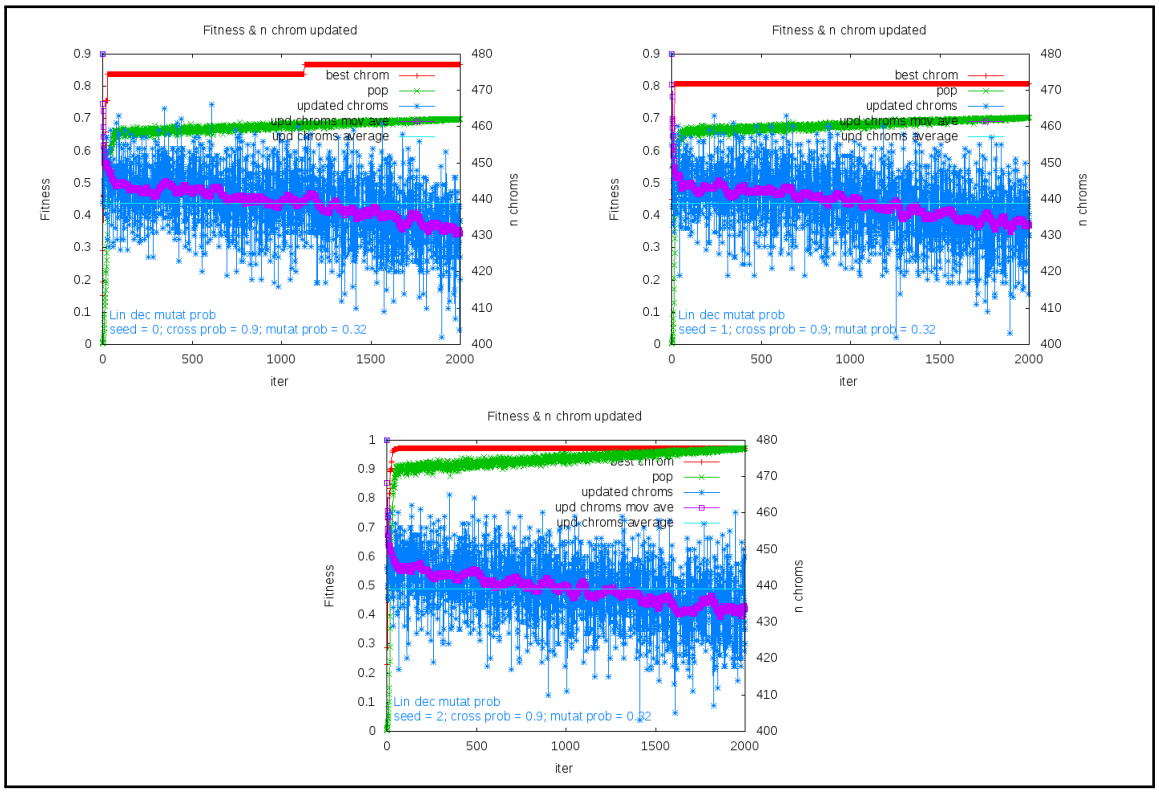


Figure A.84: Init mutation probability 0.32

Bibliography

[Adorio and Diliman, 2005] Adorio, E. P. and Diliman, U. (2005). Mvf-multivariate test functions library in c for unconstrained global optimization. *Quezon City, Metro Manila, Philippines*, pages 100–104.

[Afacan and Dundar, 2019] Afacan, E. and Dundar, G. (2019). A comprehensive analysis on differential cross-coupled CMOS LC oscillators via multi-objective optimization. *Integration*, 67:162–169.

[Afacan et al., 2021] Afacan, E., Lourenço, N., Martins, R., and Dündar, G. (2021). Review: Machine learning techniques in analog/RF integrated circuit design, synthesis, layout, and test. *Integration*, 77:113–130.

[Allen and Holberg, 2012] Allen, P. E. and Holberg, D. R. (2012). *CMOS Analog Circuit Design*. Oxford University Press, Inc, 3rd edition.

[Ando and Iba, 2000] Ando, S. and Iba, H. (2000). Analog circuit design with a variable length chromosome. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 994–1001. IEEE.

[Arora and Sinha, 2013] Arora, A. and Sinha, M. (2013). Applying variable chromosome length genetic algorithm for testing dynamism of web application. In *Proc. Int. Conf. Recent Trends in Information Technology (ICRTIT)*, pages 539–545.

[Barros et al., 2010] Barros, M., Guilherme, J., and Horta, N. (2010). Analog circuits optimization based on evolutionary computation techniques. *INTEGRATION, the VLSI journal*, 43(1):136–155.

[Beaulieu et al., 2023] Beaulieu, P.-O., Dumesnil, E., Nabki, F., and Boukadoum, M. (2023). Analog RF Circuit Sizing by a Cascade of Shallow Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(12):4391–4401.

[Bechikh et al., 2017] Bechikh, S., Datta, R., and Gupta, A., editors (2017). *Recent Advances in Evolutionary Multi-objective Optimization*, volume 20 of *Adaptation, Learning, and Optimization*. Springer.

[Box, 1957] Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101.

[Bradford et al., 1996] Bradford, N., Buttlar, D., and Farrell, J. P. (1996). *Pthreads programming*. O'Reilly & Associates, Inc. Editor: Andy Oram.

[Bremermann, 1962] Bremermann, H. J. (1962). Optimization through evolution and recombination. In Yovitis, M. C. and Jacobi, G. T., editors, *Self-Organizing Systems*, pages 93–106. Spartan, Washington, D.C.

[Brie and Morignot, 2005] Brie, A. H. and Morignot, P. (2005). Genetic planning using variable length chromosomes. In *ICAPS*, pages 320–329.

[Campilho-Gomes, 2014] Campilho-Gomes, M. (2014). Initial tests in thread-safe programming in C and in load distribution policies. Internal research report carried out within the PhD Program in Electrical and Computer Engineering of the Faculty of Sciences and Technology of the New University of Lisbon.

[Campilho-Gomes et al., 2020] Campilho-Gomes, M., Tavares, R., and Goes, J. (2020). Automatic Flat-Level Circuit Generation with Genetic Algorithms. In *Technological Innovation for Life Improvement: 11th IFIP WG 5.5/SOCOLNET Advanced Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2020*, volume Proceedings 11, pages 101–108, Costa de Caparica, Portugal. Springer.

[Campilho-Gomes et al., 2024] Campilho-Gomes, M., Tavares, R., and Goes, J. (2024). Analog flat-level circuit synthesis with genetic algorithms. *IEEE Access*, 12:115532–115545.

[Canelas et al., 2020] Canelas, A., Póvoa, R., Martins, R., Lourenço, N., Guilherme, J., Carvalho, J. P., and Horta, N. (2020). FUZYE: A Fuzzy C-Means Analog IC Yield Optimization Using Evolutionary-Based Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(1):1–13.

[Carley et al., 1996] Carley, L. R., Gielen, G. G. E., Rutenbar, R. A., and Sansen, W. (1996). Synthesis tools for mixed-signal ICs: Progress on frontend and backend strategies. In *Proceedings of the 33rd annual Design Automation Conference*, pages 298–303. ACM.

[Cavill et al., 2006] Cavill, R., Smith, S. L., and Tyrrell, A. M. (2006). Variable length genetic algorithms with multiple chromosomes on a variant of the onemax problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1405–1406, New York, NY, USA. ACM.

[Coello, 2002] Coello, C. A. C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12):1245–1287.

[Corp, 2015] Corp, S. R. (2015). Analog Market: Making Digital Systems Come Alive. https://semico.com/content/analog-market-making-digital-systems-come-alive. Accessed: 2018-07-20.

[Datta et al., 2019] Datta, D., Ray, B., and Banerjee, A. (2019). Synthesis of Linear and Non-linear Analog Circuits. In *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, pages 193–194.

[Deb, 2001] Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. Wiley.

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[Deb et al., 2007] Deb, K., Sindhya, K., and Okabe, T. (2007). Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1187–1194, New York, NY, USA. ACM.

[Deif and Gadallah, 2014] Deif, D. S. and Gadallah, Y. (2014). Wireless sensor network deployment using a variable-length genetic algorithm. In *Proc. IEEE Wireless Communications and Networking Conf. (WCNC)*, pages 2450–2455.

[Dendouga and Oussalah, 2015] Dendouga, A. and Oussalah, S. (2015). Two stage CMOS operational transconductance amplifier for front-end electronics design using multiobjective genetic algorithms. In *Proc. Signals Devices (SSD15) 2015 IEEE 12th Int. Multi-Conf. Systems*, pages 1–5.

[Doboli and Umbarkar, 2014] Doboli, A. and Umbarkar, A. (2014). The role of precedents in increasing creativity during iterative design of electronic embedded systems. *Design Studies*, 35(3):298–326.

[Doboli and Vemuri, 2003] Doboli, A. and Vemuri, R. (2003). Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(11):1556–1568.

[Domingues et al., 2022] Domingues, J., Gusmão, A., Horta, N., Lourenço, N., and Martins, R. (2022). Accelerating Voltage-Controlled Oscillator Sizing Optimizations with ANN-based Convergence Classifiers and Frequency Guess Predictors. In *2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–4.

[El-Turky and Perry, 1989] El-Turky, F. and Perry, E. E. (1989). BLADES: an artificial intelligence approach to analog circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):680–692.

[Faragó et al., 2014] Faragó, C., Lodin, A., and Groza, R. (2014). An operational transconductance amplifier sizing methodology with genetic algorithm-based optimization. *Acta Technica Napocensis. Electronica-Telecomunicatii*, 55(1):15–20.

[Farago et al., 2015] Farago, C., Oltean, G., Farago, P., and Hintea, S. (2015). An evolutionary approach for nominal design and yield enhancement of analog amplifiers. In *Proc. IEEE 10th Jubilee Int. Symp. Applied Computational Intelligence and Informatics*, pages 265–270.

[Ferent and Doboli, 2011] Ferent, C. and Doboli, A. (2011). Measuring the uniqueness and variety of analog circuit design features. *INTEGRATION, the VLSI journal*, 44(1):39–50.

[Ferent and Doboli, 2014a] Ferent, C. and Doboli, A. (2014a). Analog circuit design space description based on ordered clustering of feature uniqueness and similarity. *Integration, the VLSI Journal*, 47(2):213–231.

[Ferent and Doboli, 2014b] Ferent, C. and Doboli, A. (2014b). Novel Circuit Topology Synthesis Method Using Circuit Feature Mining and Symbolic Comparison. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 1–4, 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

[Fonseca and Fleming, 1993] Fonseca, C. M. and Fleming, P. J. (1993). Multiobjective genetic algorithms. In *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pages 6/1–6/5.

[Forstén, 2012] Forstén, H. (2012). Evolutionary algorithms and analog electronic circuits. http://hforsten.com/evolutionary-algorithms-and-analog-electronic-circuits.html. Henrik's Blog.

[Gaffney et al., 2010] Gaffney, J., Green, D. A., and Pearce, C. E. M. (2010). Binary versus real coding for genetic algorithms: A false dichotomy? In Howlett, P., Nelson, M., and Roberts, A. J., editors, *Proceedings of the 9th Biennial Engineering Mathematics and Applications Conference, EMAC-2009*, volume 51 of *ANZIAM J.*, pages C347–C359. `http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/2776` [June 22, 2010].

[Gielen and Rutenbar, 2000] Gielen, G. G. E. and Rutenbar, R. A. (2000). Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854.

[Goh and Li, 2001] Goh, C. and Li, Y. (2001). GA automated design and synthesis of analog circuits with practical constraints. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 170–177. IEEE.

[Goh and Li, 2002] Goh, C. and Li, Y. (2002). Multi-objective synthesis of cmos operational amplifiers using a hybrid genetic algorithm. In *Proc. of the 4th Asia-Pacific Conf. on Simulated Evolution and Learning*, pages 214–218.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[Goldberg and Deb, 1991] Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

[Hakhamaneshi et al., 2019] Hakhamaneshi, K., Werblun, N., Abbeel, P., and Stojanović, V. (2019). Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8.

[Harjani et al., 1989] Harjani, R., Rutenbar, R. A., and Carley, L. R. (1989). OASYS: a framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(12):1247–1266.

[Haupt and Haupt, 2004] Haupt, R. L. and Haupt, S. E. (2004). *Practical genetic algorithms*. John Wiley & Sons Inc., 2nd edition. Editor: Hoboken.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI. Second Edition, 1992.

[Horn et al., 1994] Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence*, volume 1, pages 82–87. Citeseer.

[Hornby, 2006] Hornby, G. S. (2006). Alps: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822.

[Hutt and Warwick, 2007] Hutt, B. and Warwick, K. (2007). Synapsing Variable-Length Crossover: Meaningful Crossover for Variable-Length Genomes. *IEEE Transactions on Evolutionary Computation*, 11(1):118–131.

[Héctor et al., 2010] Héctor, H., Federico, S., and Miguel, M. (2010). Analog circuit design using genetic algorithms with fuzzy fitness function. In *Proc. IEEE ANDESCON*, pages 1–5.

[ITRS, 2019] ITRS (2019). International Technology Roadmap for Semiconductors. ITRS Web-Site: http://www.itrs2.net.

[Iwata et al., 1996] Iwata, M., Kajitani, I., Yamada, H., Iba, H., and Higuchi, T. (1996). A pattern recognition system using evolvable hardware. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature — PPSN IV*, pages 761–770, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Jamil and Yang, 2013] Jamil, M. and Yang, X.-S. (2013). A literature survey of benchmark functions for global optimization problems. *Int. J. of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194.

[Jiao and Doboli, 2015] Jiao, F. and Doboli, A. (2015). A low-voltage, low-power amplifier created by reasoning-based, systematic topology synthesis. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2648–2651.

[Jiao et al., 2015] Jiao, F., Montano, S., and Doboli, A. (2015). Knowledge-intensive, causal reasoning for analog circuit topology synthesis in emergent and innovative applications. In *Proc. Automation Test in Europe Conf 2015 Design Exhibition (DATE)*, pages 1144–1149.

[Jr et al., 2014] Jr, O. V., Crepaldi, P. C., Zoccal, L. B., and Pimenta, T. C. (2014). Synthesis of passive filter using object oriented genetic algorithm. In *Proc. 26th Int. Conf. Microelectronics (ICM)*, pages 72–75.

[Kahng and Koushanfar, 2015] Kahng, A. B. and Koushanfar, F. (2015). Evolving EDA Beyond its E-Roots: An Overview. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 247–254, Piscataway, NJ, USA. IEEE Press.

[Kajitani et al., 1996] Kajitani, I., Hoshino, T., Iwata, M., and Higuchi, T. (1996). Variable length chromosome GA for evolvable hardware. In *Proc. IEEE Int. Conf. Evolutionary Computation*, pages 443–447.

[Karaboga and Basturk, 2008] Karaboga, D. and Basturk, B. (2008). On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing*, 8(1):687–697.

[Karci et al., 2016] Karci, H., TOHUMOĞLU, G., and NACAROĞLU, A. (2016). Speciation-based genetic algorithm in analog circuit design. *Turkish Journal of Electrical Engineering & Computer Sciences*, 24(3):1022–1033.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

[Kim and de Weck, 2005] Kim, I. Y. and de Weck, O. L. (2005). Variable chromosome length genetic algorithm for progressive refinement in topology optimization. *Structural and Multidisciplinary Optimization*, 29(6):445.

[Konak et al., 2006] Konak, A., Coit, D. W., and Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: a tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007.

[Koza et al., 1997a] Koza, J. R., Bennett, F. H., Andre, D., Keane, M. A., and Dunlap, F. (1997a). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on evolutionary computation*, 1(2):109–128.

[Koza et al., 1997b] Koza, J. R., Bennett, F. H., Lohn, J., Dunlap, F., Keane, M. A., and Andre, D. (1997b). Automated synthesis of computational circuits using genetic programming. In *Proc. IEEE Int Evolutionary Computation Conf*, pages 447–452.

[Koza et al., 1997c] Koza, J. R., Bennett III, F. H., Andre, D., and Keane, M. A. (1997c). Evolution using genetic programming of a low-distortion, 96 decibel operational amplifier. In *Proceedings of the 1997 ACM symposium on Applied computing*, pages 207–216. ACM.

[Koza et al., 2000] Koza, J. R., Bennett III, F. H., Andre, D., and Keane, M. A. (2000). Synthesis of topology and sizing of analog electrical circuits by means of genetic programming. *Computer methods in applied mechanics and engineering*, 186(2-4):459–482.

[Koza et al., 1992] Koza, J. R., Koza, J. R., and Rice, J. P. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. A Bradford book. MIT Press.

[Koza et al., 2008] Koza, J. R., Streeter, M. J., and Keane, M. A. (2008). Routine high-return human-competitive automated problem-solving by means of genetic programming. *Information Sciences*, 178(23):4434–4452.

[Krasnicki et al., 1999] Krasnicki, M., Phelps, R., Rutenbar, R., and Carley, L. (1999). Maelstrom: efficient simulation-based synthesis for custom analog cells. In *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pages 945–950.

[Kruiskamp, 1995] Kruiskamp, D. L. W. (1995). DARWIN: CMOS opamp synthesis by means of a genetic algorithm. In *Proc. 32nd Design Automation Conf*, pages 433–438.

[Linden and Somaya, 2003] Linden, G. and Somaya, D. (2003). System-on-a-chip integration in the semiconductor industry: Industry structure and firm strategies. *Industrial and Corporate Change*, 12:545–576.

[Liu and He, 2012] Liu, M. and He, J. (2012). A multi-algorithm framework for the evolvement of analog circuits. In *Proc. Int. Conf. Systems and Informatics (ICSAI2012)*, pages 780–784.

[Lohn and Colombano, 1998] Lohn, J. D. and Colombano, S. P. (1998). Automated analog circuit synthesis using a linear representation. In Moshe Sipper, Daniel Mange, A. P.-U., editor, *International Conference on Evolvable Systems*, pages 125–133, Lausanne, Switzerland. Springer, Springer Berlin, Heidelberg.

[Lohn et al., 2000] Lohn, J. D., Colombano, S. P., Haith, G. L., Stassinopoulos, D., and Norvig, P. (2000). A parallel genetic algorithm for automated electronic circuit design. In *Proc. Computational Aerosciences Workshop*.

[Lourenço et al., 2016] Lourenço, N., Martins, R., Canelas, A., Póvoa, R., and Horta, N. (2016). AIDA: Layout-aware analog circuit-level sizing with in-loop layout generation. *Integration*, 55:316–329.

[Maji et al., 2016] Maji, K. B., Kar, R., Mandal, D., and Ghoshal, S. P. (2016). An evolutionary approach based design automation of low power CMOS Two-Stage Comparator and Folded Cascode OTA. *AEU-International Journal of Electronics and Communications*, 70(4):398–408.

[Martens and Gielen, 2008] Martens, E. S. J. and Gielen, G. (2008). *High-level modeling and synthesis of analog integrated systems*. Springer Science & Business Media.

[Martins et al., 2012] Martins, R., Lourenço, N., and Horta, N. (2012). Multi-Objective Multi-Constraint Routing of Analog ICs using a Modified NSGA-II Approach. In *Proc. Analysis and Simulation Methods and Applications to Circuit Design (SMACD) 2012 Int. Conf. Synthesis, Modeling*, pages 65–68.

[Martins et al., 2017] Martins, R. M., Lourenço, N., and Horta, N. (2017). *Analog Integrated Circuit Design Automation – Placement, Routing and Parasitic Extraction Techniques*. Springer International Publishing.

[Mattiussi and Floreano, 2007] Mattiussi, C. and Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, 11(5):596–607.

[McConaghy et al., 2009] McConaghy, T., Palmers, P., Steyaert, M., and Gielen, G. G. E. (2009). Variation-aware structural synthesis of analog circuits via hierarchical building blocks and structural homotopy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(9):1281–1294.

[McGrath, 2018] McGrath, D. (2018). Analog Seen as Fastest-Growing Chip Segment. https://www.eetimes.com/. Accessed: 2018-07-20.

[Mentor Graphics, 2009] Mentor Graphics (2009). Eldo Platform. http://www.mentor.com/. Eldo release AMS 2009.2.

[Mydlowec and Koza, 2000] Mydlowec, W. and Koza, J. (2000). Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 187–197, Las Vegas, Nevada.

[Mysore et al., 2006] Mysore, G. D., Conrad, J. M., and Newberry, B. (2006). A microcontroller-based bed-of-nails test fixture to program and test small printed circuit boards. In *Proceedings of the IEEE SoutheastCon 2006*, pages 104–107. IEEE.

[Ni et al., 2016] Ni, J., Wang, K., Huang, H., Wu, L., and Luo, C. (2016). Robot path planning based on an improved genetic algorithm with variable length chromosome. In *Proc. Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) 2016 12th Int. Conf. Natural Computation*, pages 145–149.

[Nicosia et al., 2008] Nicosia, G., Rinaudo, S., and Sciacca, E. (2008). An evolutionary algorithm-based approach to robust analog circuit design using constrained multi-

objective optimization. In *Research and Development in Intelligent Systems XXIV*, pages 7–20. Springer, London, UK.

[Nourani and Andresen, 1998] Nourani, Y. and Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373.

[Obaidullah and Khan, 2017] Obaidullah, M. and Khan, G. N. (2017). Hybrid multi-swarm optimization based NoC synthesis. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 62–67.

[Passos et al., 2018] Passos, F., Martins, R., Lourenco, N., Roca, E., Castro-López, R., Povoa, R., Canelas, A., Horta, N., and Fernández, F. V. (2018). Handling the effects of variability and layout parasitics in the automatic synthesis of lnas. In *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–164. IEEE.

[Passos et al., 2019] Passos, F., Roca, E., Castro-López, R., Horta, N., and Fernandez, F. V. (2019). Synthesis of mm-wave circuits using-em-simulated passive structure libraries. In *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 57–60. IEEE.

[Patel, 2014] Patel, T. (2014). Comparison of Level 1, 2 and 3 MOSFET's. techreport, University of Texas at Arlington.

[Paulino et al., 2001] Paulino, N., Goes, J., and Steiger-Garcao, A. (2001). Design methodology for optimization of analog building blocks using genetic algorithms. In *Proc. ISCAS 2001. The 2001 IEEE Int. Symp. Circuits and Systems (Cat. No.01CH37196)*, volume 5, pages 435–438 vol. 5.

[Pawar and Bichkar, 2015] Pawar, S. N. and Bichkar, R. S. (2015). Genetic algorithm with variable length chromosomes for network intrusion detection. *International Journal of Automation and Computing*, 12(3):337–342.

[Phelps et al., 2000] Phelps, R., Krasnicki, M., Rutenbar, R., Carley, L., and Hellums, J. (2000). Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(6):703–717.

[Rajasekaran and Pai, 2017] Rajasekaran, S. and Pai, G. A. V. (2017). *Neural Networks, Fuzzy Systems and Evolutionary Algorithms: Synthesis and Applications*. PHI Learning Pvt. Ltd.

[Ratnesh et al., 2021] Ratnesh, R., Goel, A., Kaushik, G., Garg, H., Chandan, Singh, M., and Prasad, B. (2021). Advancement and challenges in mosfet scaling. *Materials Science in Semiconductor Processing*, 134:106002.

[Rocha et al., 2014] Rocha, F. A. E., Martins, R. M. F., Lourenço, N. C. C., and Horta, N. C. G. (2014). *Electronic Design Automation of Analog ICs combining Gradient Models with Multi-Objective Evolutionary Algorithms*. Springer Science & Business Media.

[Sabat et al., 2009] Sabat, S. L., Kumar, K., and Udgata, S. K. (2009). Differential evolution and swarm intelligence techniques for analog circuit synthesis. In *2009 World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 469–474.

[Salvador, 2016] Salvador, R. (2016). Evolvable hardware in fpgas: Embedded tutorial. In *Proc. Int. Conf. Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6.

[Santos-Tavares, 2010] Santos-Tavares, R. (2010). *Time-domain optimization of amplifiers based on distributed genetic algorithms*. phdthesis, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Lisbon.

[Santos-Tavares et al., 2008] Santos-Tavares, R., Paulino, N., Higino, J., Goes, J., and Oliveira, J. P. (2008). Optimization of multi-stage amplifiers in deep-submicron CMOS using a distributed/parallel genetic algorithm. In *Proc. IEEE Int. Symp. Circuits and Systems*, pages 724–727.

[Sapargaliyev and Kalganova, 2006a] Sapargaliyev, Y. and Kalganova, T. (2006a). Absolutely free extrinsic evolution of passive low-pass filter. In *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on*, pages 1210–1213. IEEE.

[Sapargaliyev and Kalganova, 2006b] Sapargaliyev, Y. and Kalganova, T. (2006b). Constrained and unconstrained evolution of "lcr" low-pass filters with oscillating length representation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1529–1536. IEEE.

[Sapargaliyev and Kalganova, 2006c] Sapargaliyev, Y. and Kalganova, T. (2006c). Unconstrained evolution of close-to-ideal " lcr" low-pass filter. In *2006 International Conference on Intelligent Engineering Systems*, pages 145–150.

[Sapargaliyev and Kalganova, 2010a] Sapargaliyev, Y. and Kalganova, T. G. (2010a). Automated synthesis of 8-output voltage distributor using incremental, evolution. In *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, pages 186–193. IEEE.

[Sapargaliyev and Kalganova, 2010b] Sapargaliyev, Y. A. and Kalganova, T. G. (2010b). Challenging the evolutionary strategy for synthesis of analogue computational circuits. *Journal of Software Engineering and Applications*, 3(11):1032–1039.

[Schaffer, 1985] Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers.

[Sedra and Smith, 2007] Sedra, A. S. and Smith, K. C. (2007). *Microelectronic Circuits Revised Edition*. Oxford University Press, Inc., USA, 5th edition.

[Sedra and Smith, 2014] Sedra, A. S. and Smith, K. C. (2014). *Microelectronic Circuits*. Oxford University Press.

[Serra, 2017] Serra, H. A. (2017). *Analysis and Design Methodologies for Switched-Capacitor Filter Circuits in Advanced CMOS Technologies*. phdthesis, Universidade Nova de Lisboa, FCT.

[Sorkhabi and Zhang, 2017] Sorkhabi, S. E. and Zhang, L. (2017). Automated topology synthesis of analog and rf integrated circuits: A survey. *Integration*, 56:128–138.

[SourceForge, 2019] SourceForge (2019). Mixed Mode-Mixed Level Circuit Simulator based on Berkeley's SPICE 3F5. http://ngspice.sourceforge.net/. Ver. 31.

[Sripramong and Toumazou, 2002] Sripramong, T. and Toumazou, C. (2002). The invention of cmos amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1237–1252.

[Storn and Price, 1997] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

[Stringer, 2007] Stringer, H. L. (2007). Behavior of variable-length genetic algorithms under random selection. mathesis, University of Central Florida, School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science.

[SYNOPSYS, 2010] SYNOPSYS (2010). *HSPICE® Reference Manual: MOSFET Models*. SYNOPSYS.

[Synopsys, 2017] Synopsys (2017). HSPICE. https://www.synopsys.com/. Ver. M-2017.03.

[Tavares et al., 2003] Tavares, R., Vaz, B., Goes, J., Paulino, N., and Steiger-Garcao, A. (2003). Design and optimization of low-voltage two-stage CMOS amplifiers with enhanced performance. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 1, pages I–197–I–200 vol.1.

[Thompson and Layzell, 1999] Thompson, A. and Layzell, P. (1999). Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79.

[Tomassini, 1995] Tomassini, M. (1995). A survey of genetic algorithms. In *Annual reviews of computational physics III*, pages 87–118. World Scientific.

[Torresen, 1998] Torresen, J. (1998). A divide-and-conquer approach to evolvable hardware. In Sipper, M., Mange, D., and Pérez-Uribe, A., editors, *Evolvable Systems: From Biology to Hardware*, pages 57–65, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Torresen, 2004] Torresen, J. (2004). An Evolvable Hardware Tutorial. *Field Programmable Logic and Application*.

[Trefzer, 2006] Trefzer, M. A. (2006). *Evolution of transistor circuits*. PhD thesis, Ruperto-Carola-University of Heidelberg, Heidelberg, Germany.

[Ushie et al., 2017] Ushie, O. J., Abbod, M. F., and Ogbulezie, J. C. (2017). The use of Genetic Programming to Evolve Passive Filter Circuits. *International Journal of Engineering and Technology Innovation*, 7(4):255–268.

[Vaz et al., 2001] Vaz, B., Costa, R., Paulino, N., Goes, J., Tavares, R., and Steiger-Garcao, A. (2001). A general-purpose kernel based on genetic algorithms for optimization of complex analog circuits. In *Proc. 44th IEEE 2001 Midwest Symp. Circuits and Systems. MWSCAS 2001 (Cat. No.01CH37257)*, volume 1, pages 83–86 vol.1.

[Wang et al., 2008] Wang, F., Li, Y., Li, K., and Lin, Z. (2008). A new circuit representation method for analog circuit design automation. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1976–1980.

[Whitley, 1994] Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.

[Whitley et al., 1989] Whitley, L. D. et al. (1989). The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, volume 89, pages 116–123. Fairfax, VA.

[Williams, 1998] Williams, J. (1998). *The art and science of analog circuit design*. Newnes. Editor: Elsevier Science Publishers.

[Wright et al., 2003] Wright, R., Zgol, M., Adebimpe, D., and Kirkland, L. (2003). Functional circuit board testing using nanoscale sensors. In *Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference.*, pages 266–272. IEEE.

[Yoon and Kim, 2012] Yoon, Y. and Kim, Y.-H. (2012). *Bio-Inspired Computational Algorithms and Their Applications*, chapter The Roles of Crossover and Mutation in Real-Coded Genetic Algorithms, pages 65–82. InTech. Editor: Dr. Shangce Gao.

[Yuan and He, 2010] Yuan, H. and He, J. (2010). Evolutionary design of operational amplifier using variable-length differential evolution algorithm. In *Proc. Int. Conf. Computer Application and System Modeling (ICCASM 2010)*, volume 4, pages 610–614.

[Zebulum et al., 1998] Zebulum, R. S., Pacheco, M. A., and Vellasco, M. (1998). Comparison of different evolutionary methodologies applied to electronic filter design. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 434–439. IEEE.

[Zebulum et al., 2000] Zebulum, R. S., Vellasco, M., and Pacheco, M. A. (2000). Variable Length Representation in Evolutionary Electronics. *Evolutionary Computation*, 8(1):93–120.

[Zhang et al., 2007] Zhang, J., Chung, H. S. H., and Lo, W. L. (2007). Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 11(3):326–335.

[Zhang and Li, 2007] Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731.

[Zhao and Zhang, 2020] Zhao, Z. and Zhang, L. (2020). An automated topology synthesis framework for analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4325–4337.

[Žiga Rojec et al., 2019] Žiga Rojec, Árpád Bűrmen, and Fajfar, I. (2019). Analog circuit topology synthesis by means of evolutionary computation. *Engineering Applications of Artificial Intelligence*, 80:48–65.