ChatGPT

Peer Review of *"Producing First Draft of Go Code from Unit-Tests with Genetic Programming"*

Writing Quality (Turkish & English)

- **English Abstract Grammar:** The English abstract contains numerous grammatical errors and awkward phrasing that obscure the meaning. For example, it states *"there have been no success on applying it to an imperative and compiled language"* [1], which is not proper English (it should be "no success in applying it..."). Similarly, *"whenever a candidate with runtime, syntax or print errors is appeared in the main search"* is ungrammatical. Such errors, including incorrect comparatives like "**worser** ones" [1], suggest the English sections were not proofread by a fluent speaker. These issues reduce clarity and professionalism. The abstract's run-on sentences are difficult to follow, e.g. a single sentence spans multiple lines describing how rich language features expand the solution space beyond what evolutionary search can handle [2]. Breaking these into clear, concise sentences would improve comprehension.
- **Turkish Text Clarity:** The Turkish portions of the thesis are written in formal academic language but often suffer from very long, complex sentences that can challenge the reader. For instance, the Turkish summary sentence explaining why GP hasn't been adapted to modern languages extends over four lines [3] with multiple clauses. While grammatically correct, such lengthy sentences impede quick understanding. The writing could be made more concise – splitting complex thoughts into shorter sentences and using clearer subject–predicate structures. On the positive side, technical terminology is used consistently in Turkish (e.g. *denek* for individual, *başarım* for fitness), but the text sometimes reads as overly dense. Some explanations feel circuitous or repetitive, which affects readability.
- **Jargon and Terminology:** The thesis introduces a lot of specialized terminology (GP concepts, evolutionary terms, and new method names like *Katmanlı Arama* and *ST-ASTGP*). In Turkish, these are generally defined, but the heavy use of jargon can overwhelm the reader. For example, terms like *fenotip*, *AST*, *tip güvenliği* appear frequently; readers not already versed in genetic programming might struggle, even though a glossary is provided. The English abstract also uses uncommon phrases (e.g. "print errors" for code generation errors) that might confuse readers [2]. More reader-friendly definitions or examples would help, especially since the work spans two languages.
- **Conciseness and Redundancy:** The writing tends to be verbose. Some sections reiterate points or include superfluous commentary. For instance, the introduction is subdivided into *Yöntem, Amaç, Önem* (Method, Aim, Significance) but these sections collectively repeat motivations that could be stated once. The discussion chapter also restates background concepts (like comparing evolutionary algorithms to biological evolution) that were already explained in earlier chapters, adding little new insight. Eliminating such redundancy and focusing text on the core contributions would strengthen the narrative. Overall, tightening the prose – both in Turkish and English – would greatly improve readability.

Technical Correctness

- **Understanding of GP and Algorithms:** The thesis demonstrates a solid understanding of genetic programming fundamentals and related algorithms. It correctly explains evolutionary algorithm concepts (e.g. selection, mutation, crossover) and known challenges like local minima and bloat. The descriptions of strongly-typed GP (STGP) and AST-based GP are technically sound, and the author is aware of why naive AST-based genetic programming massively expands the search space [4]. There is no evident misrepresentation of standard GP theory; foundational methods (Koza's approach, ADFs, etc.) are explained accurately.
- **Proposed Method – *ST-ASTGP*:** The *Strongly-Typed AST Genetic Programming (ST-ASTGP)* method is logically motivated and mostly technically correct. Enforcing type safety in both the genotype (tree nodes) **and** phenotype (actual Go code) is a sensible approach to reduce invalid programs. The thesis correctly identifies that many AST-generated individuals fail due to type mismatches or syntax errors, and thus extends STGP concepts to the AST domain. The implementation leverages Go's `go/types` for compile-time type checking, which is an appropriate choice [5]. There is no indication of flawed logic in how ST-ASTGP works – it should indeed produce fewer syntactically invalid mutants. However, one concern is that the thesis somewhat implies ST-ASTGP nearly solves the syntax problem, whereas in reality it *reduces* it but doesn't eliminate all compile errors (e.g. it can't prevent logic errors or all semantic issues). The text could be more precise here. For example, Appendix results show 28 out of 200 offspring were syntactically valid without ST-ASTGP [6] [7] – implying ST-ASTGP would raise that number, but the thesis doesn't quantify by how much. It's technically correct that type-safe mutations improve success rate, but the work stops short of fully proving how much improvement is gained (no comparative stats are given).
- **Proposed Method – *Katmanlı Arama (Layered Search):*** The *Layered Search* algorithm is an innovative multi-layer evolutionary scheme. Technically, the design is plausible: when an individual fails at some test stage (printing, compiling, or runtime), a restricted "sub-evolution" is spawned to specifically fix that level of error [8] [4]. These sub-evolutions run for a limited depth/generations to attempt *n*-step repairs, after which the candidate is returned to the main pool if improved. The mechanism is described with pseudocode for four search layers (main search and three nested searches for program-level, code-level, AST-level candidates) [9] [10]. This design is logically consistent and addresses the noted issue of "premature elimination" of individuals that need multiple mutations to become viable. On paper, the algorithm makes sense. A potential technical concern, however, is complexity and parameter tuning: spawning many sub-searches could explode the search workload if not carefully limited. The thesis does mention generation limits for sub-searches [11], but it doesn't deeply analyze the computational overhead. Technically, there's also a risk that focusing on fixing syntax/runtime errors might isolate those fixes but not guarantee the fixed individual will then pass the semantic (unit test) stage – the approach assumes separate concerns, which may hold generally, but some fixes might interact. The thesis does not discuss such edge cases. Still, no outright errors are apparent in the algorithm's logic, though its efficiency and completeness are not fully validated.
- **Validity of Methods:** The methods are conceptually sound given the problem context. The author correctly identifies known pitfalls in evolving code for compiled languages (e.g. the huge search space, long compilation times, side effects) and proposes reasonable solutions. The use of actual unit tests as a fitness measure is appropriate and aligns with prior work (GP-based program repair). The fitness evaluation scheme, assigning different fitness tiers for syntax errors vs runtime errors vs test failures, is a valid way to juggle multi-criteria success [12] [13]. I found no fundamental flaws in how fitness is calculated or how genetic operators are applied. One minor issue: the thesis asserts *"even after 30 years, GP has had no success on imperative, compiled languages"* [2] – while essentially

true in the sense of *general-purpose* success, this statement is slightly exaggerated. There have been small-scale successes (e.g. GP produced certain algorithms or repairs in C code [14] [15]), albeit not broad or industrial. The thesis uses this claim to motivate the work, which is fine, but a more nuanced acknowledgement of prior partial successes would be technically accurate. Overall, the technical content of the thesis is correct in its algorithms and understanding – the weaknesses lie more in demonstrating those algorithms' effectiveness rather than in their formulation.

Structure and Organization

- **Overall Structure:** The thesis follows a conventional structure for a master's thesis and is mostly well-organized. It opens with an introduction (including method, aim, and significance), then provides extensive background (Concepts and Past Work), describes the Tools and Methodology, presents Results (Findings), and concludes with Discussion, Recommendations, and Conclusion. This logical flow from fundamentals to implementation to results is appropriate. Each chapter has a clear purpose, and the table of contents is detailed and helpful. There is a good separation between general background (Chapter 2), literature review (Chapter 3), and the author's contributions (Chapters 5 and 6).
- **Coherence Between Chapters:** Generally, chapters transition sensibly. The introduction sets up the problem of evolving Go code from tests, highlighting challenges that later chapters address. The background chapters (2 and 3) provide necessary context on evolutionary algorithms, GP variants, and prior attempts, which feed directly into the rationale for ST-ASTGP and Layered Search in Chapter 5. One critique is that Chapter 7 (Discussion) drifts somewhat from the specific results into a broad comparison of evolutionary algorithms and biological evolution [16] [17] and musings on GP's applicability. This content, while interesting, feels disconnected from the immediate findings of this thesis. It reiterates concepts like *GP mutations being random and often counterproductive without guidance*, which were already covered when introducing ST-ASTGP and CA-STG [17]. This makes the discussion chapter less focused. A stronger structure would tie the discussion directly to how the results met (or didn't meet) the thesis objectives, rather than rehashing general observations.
- **Balance of Sections:** The thesis devotes a lot of space to background (the Concepts chapter is quite lengthy, covering everything from basic evolutionary strategies to TDD). This ensures the work is self-contained, but some parts of Chapter 2 could be trimmed. For example, explaining generic evolutionary strategies or local vs global minima at length, when the target audience likely knows these or when they could be summarized in a few lines, slightly dilutes focus [18] [19]. In contrast, the core contributions in Chapter 5 and the results in Chapter 6 feel concise. It might have been better to allocate more space in results/discussion to analyzing the performance of the new methods, and less on elemental concepts. Chapter 8 (Recommendations/Future Work) is a useful addition, showing coherence by reflecting on what was learned and what can be improved or tried next – it connects well with the challenges described earlier.
- **Flow and Redundancy:** Apart from the discussion chapter's issues, the flow within chapters is logical. The Methodology chapter (5) systematically goes through each component of the approach (from parsing unit tests, building the symbol table, mutation/crossover, to the introduction of ST-ASTGP and Layered Search, and finally fitness evaluation and selection). This step-by-step breakdown is easy to follow. There is minor redundancy: the introduction's *"Aim"* vs *"Significance"* sections overlap – both justify why the problem is important. They could likely be merged to streamline the intro. Similarly, some points in the literature review (Chapter 3) echo earlier definitions (e.g. re-introducing STGP or bloat control which were touched on in Chapter 2). These are small organizational niggles. Overall, the thesis is organized in a standard and mostly effective manner, with clear chapter

demarcations. Tightening the connections in the discussion and cutting a bit of repeated exposition would improve coherence.

Originality and Contribution

- **Novelty of *Layered Search*:** The concept of *Katmanlı Arama* (Layered Search) appears to be a novel contribution of this thesis. In the genetic programming literature, it's uncommon to spawn hierarchical evolutionary processes to handle different "failure modes" of individuals on the fly. This idea of isolating candidates that, for example, don't compile, and evolving them in a special sandbox until they do (then returning them to the main population) is quite original. It tackles the unique multi-stage nature of compiled code synthesis (AST validity, compilability, executability, correctness) in a creative way. While the thesis references *Layered Learning* in AI [20], the Layered Search here is distinct in implementation. This is a significant conceptual contribution – it provides a possible solution to premature convergence caused by strict survival criteria at early stages (syntax/runtime). The significance could be high for program synthesis GP, as it offers a pathway to evolve solutions that require a sequence of correlated changes (something standard GP struggles with). That said, the thesis somewhat overstates its novelty in isolation; one could view Layered Search as an engineered workaround to known GP limitations rather than a fundamentally new evolutionary paradigm. It's essentially a sophisticated form of adaptive evolution control. Nonetheless, it is a fresh approach in this domain and a clear contribution of the work.
- **Novelty of *ST-ASTGP*:** Applying strong typing to AST-based GP (ST-ASTGP) is a more incremental contribution, but still noteworthy. Strongly Typed GP itself is not new (dates to Montana 1995), and AST-based GP has been explored; however, combining them in the context of evolving real Go code from unit tests is novel in practice. The thesis introduces *CA-STG (Context-Aware Subtree Generation)* as part of ST-ASTGP to ensure any newly generated AST subtree respects Go's type rules [21]. This goes beyond typical STGP by using the language's actual type checker. The result is fewer malformed programs. This contribution is somewhat technical/implementation-oriented, but important for making GP work on a real programming language. It's not a flashy, theoretical innovation, but a necessary advancement to push GP into imperative language synthesis. The originality here lies in the execution details (leveraging Go's compiler APIs within GP) more than in theory. It closes a gap in literature where many assumed GP on full ASTs would be intractable – the thesis shows a viable path forward.
- **Significance and Impact:** The thesis's contributions are in a niche but potentially impactful area: evolutionary code synthesis for compiled languages. The work is timely, given renewed interest in code generation. If layered evolutionary search and ST-ASTGP can even partially overcome the long-standing barriers (search space explosion, compile overhead, etc.), that's a significant step for GP research. However, the actual impact is a bit unclear from the thesis because the contributions are demonstrated on a limited scale. They are presented as concept validations rather than proven breakthroughs. For example, we are told Layered Search *"aims to protect explorative character of search"* and *gives multi-step innovations a chance*, but we aren't shown a concrete problem that couldn't be solved before and now can be thanks to this method. Thus, the contributions, while original, currently feel **preliminary**. The thesis itself acknowledges it's sharing "solution ideas" to spark further research, which suggests the contributions are more conceptual groundwork. In summary: **Originality is decent** (especially Layered Search), but the **significance is limited by lack of demonstrated success**. Future work building on these ideas will determine how important they become.

- **Avoiding Overstatement:** Generally, the author does not grossly overclaim credit – the tone is that these are proposed methods to help GP, not that they have *solved* code synthesis. One area of mild overstatement is in the discussion where it's *predicted that GP will outshine other ML methods in solving unique programming problems* [22]. This broad claim isn't really backed up by the results in the thesis (at this point it's aspirational). It doesn't detract from the contributions per se, but it's a forward-looking statement that might be too optimistic. A more measured summary of contributions would be that the thesis introduces two novel techniques to improve GP-based program synthesis and provides a groundwork for future studies, rather than implying GP will now be the superior choice for unique coding problems.

Results and Discussion

- **Experimental Setup Adequacy:** The experimental evaluation in the thesis is unfortunately one of its weaker aspects. The thesis does not describe a clear set of target programming problems (unit tests) that were attempted and solved or not solved. It's therefore hard to gauge how well the approach actually works. We see that the author measured things like the proportion of syntactically correct individuals, the effect of using in-memory compilation to reduce compile time, etc. For example, compile-time benchmarking with a "Hello World" program and a "Terraform" program shows a 19% speed improvement by using a RAM disk for the compiler [23] [24]. While useful, these are **system-level metrics** rather than end-goal metrics. The thesis lacks a direct demonstration like: "Using our GP system, we successfully evolved a program that passes the given unit tests (or solved a known benchmark problem)." It appears that none of the runs actually produced a completely correct program that passed all tests – at least, no such success is explicitly reported. This is a critical omission: without that, the results mostly show that the methods *could* make GP more efficient or plausible, but not that they achieved the end result.
- **Data Analysis and Findings:** The findings presented focus on intermediate outcomes. In Chapter 6, results are organized by ASTGP vs STGP vs ST-ASTGP, and compile time. The thesis notes, for instance, that ST-ASTGP generates far fewer syntax errors than plain ASTGP (implicitly shown by an example where only 14% of children were valid without ST-ASTGP [6] [25]). It also qualitatively argues that Layered Search prevents premature loss of useful genetic material – e.g. the discussion points out that an unguided mutation operator will introduce small errors when trying to fix bigger issues, and that a diversity-preserving selection alone can't solve this because some lineages need much more time/depth [17] [26]. These arguments are sensible and consistent with the design of the methods. However, they remain *qualitative*. The thesis would benefit from quantitative comparisons – for example: *"With Layered Search enabled, the GP found a solution in X% of runs for a given test suite, whereas without it, it never did"* or *"ST-ASTGP reduced the average number of compiler errors per generation from N to M, and cut down the generations needed to get a runnable program by Y%."* Such data is absent. The only numerical results are the compile time reductions (19% improvement using tmpfs) [23] and a table of typical error types vs estimated mutations needed (which is more of a theoretical estimate than experimental data). The lack of clear success/failure metrics or charts (e.g. fitness over generations) is a major weakness in the results chapter. In summary, the experiments seem *limited and somewhat informal*. They demonstrate that the system runs and that the new techniques do *something plausible*, but they do not strongly support the claim that *"GP can produce a first draft of Go code from unit tests."*
- **Discussion Quality:** The discussion chapter (Chapter 7) is lengthy but doesn't directly critique the results in a frank way. Rather than analyzing why, for example, a fully correct solution wasn't found or what the success rate was, the discussion talks about GP's general strengths and weaknesses,

future prospects, and analogies to biology. Some of these points are insightful (such as acknowledging GP is computationally expensive but maybe acceptable if hardware is cheaper than human labor [27] [28]). Yet, the discussion largely sidesteps the fact that the thesis did *not* actually demonstrate a solved example. A more candid discussion of the results — e.g. *"we did not manage to evolve a completely correct program within our generation limits; likely reasons are X and Y; Layered Search helped in these ways but was not sufficient in the time frame"* — is missing. Instead, the author adopts an optimistic tone that given these new methods, GP *will eventually* get there. This comes across as somewhat **speculative**. For instance, the claim that GP can be combined with LLMs, using GP only for the truly novel parts of a problem [29] [30], is forward-looking and interesting, but it isn't derived from the thesis's experiments – it's more like future vision brainstorming. In a peer review, I'd say the discussion could be improved by directly tying back to the thesis objectives and results: Was the aim achieved? What remained unsolved? What evidence supports the solutions proposed? Currently, the discussion does not fully confront the gap between proposed method and proven result, which slightly undermines the scientific rigor.

- **Experimental Rigor:** As a whole, the evaluation feels more like a proof-of-concept demonstration than a thorough test of the hypothesis. The thesis would be stronger if it had, for example, attempted a set of small programming problems (such as functions from a program synthesis benchmark) and reported how often and how fast the GP could evolve a solution with and without the new methods. Instead, we get assurances that the methods address certain issues and anecdotal examples of candidate programs (some code listings of mutated programs are given in an appendix), but no clear measure of success. This shallow evaluation is a significant weakness – it leaves the reader unsure if the approach actually *works*. It's understandable that fully evolving correct programs in Go is extremely hard, so failure to solve a non-trivial problem isn't shameful; however, acknowledging that and analyzing partial progress would strengthen the work. In summary, the results support the plausibility of the approach (e.g. yes, it reduces compile errors and can run more iterations by speeding up compilation), but they do not yet *validate* that the GP can produce useful code. The discussion could do a better job of openly interpreting these mixed results, rather than speaking in generalities.

Figures and Tables

- **Figure Clarity and Effectiveness:** The thesis is accompanied by many figures (diagrams, charts, code snippets) and tables. In general, the figures are relevant to the content, but some are overly complex or not reader-friendly. For example, Figures 5.7 and 5.8, which illustrate how Layered Search alters the search process, are quite dense. Figure 5.8 in particular attempts to show the evolutionary path of a successful run with various intermediate forms (AST, code, program states) across a lineage [31] [32]. The description of this figure in the text is hard to parse – it refers to labels like S29 and S64, and notes how many individuals reached each state, which requires careful reading to understand. A simpler visual or a step-by-step breakdown might have been more effective. Some figures use Turkish annotations (e.g., the pseudocode in Figures 5.9–5.12 has titles like *AdayAraması.İlerle*), which is fine given the thesis language, but non-Turkish readers of the English abstract will not grasp them. In an ideal world, algorithm figures could have English keywords or a brief English description for accessibility.
- **Use of Code Examples:** There are code block figures (like the example AST in Figure 2.9 or the Go code/AST transformations in Figure 4.1) that are very helpful to illustrate concepts. The code snippet in Figure 2.11 showing a target function, and Figure 2.12 showing a unit test with multiple scenarios, effectively set the stage for readers to understand input/output of the problem [33] [34]. These are

well-chosen. In contrast, the Appendix listing of 28 mutated versions of a function (Appendix 2) is questionable in value. It literally prints out dozens of variants of `func WalkWithNils(...) { ... }` with random changes [6] [35]. While this shows what kind of bizarre code GP can generate, a few representative examples would suffice. Listing all 28 is overkill and not particularly enlightening to the reader – it feels like raw data that could be summarized. This is a minor point (as it's in appendix), but it reflects that some figures/tables weren't optimized for insight.

- **Table Effectiveness:** The tables in the thesis are mostly small and serve specific purposes. Table 5.1 and 5.2 list common runtime and compile errors in Go, along with the hypothesized number of mutations needed to fix them [36] [37]. These tables are an interesting inclusion – they give the reader an intuition of how "far away" certain errors might be from a solution (e.g., a `nil pointer` runtime error might need 1 mutation versus a logic bug needing more). However, it's not clear if these numbers are drawn from the author's experiment, from literature, or just estimated. The label says "*...estimated number of mutations...*", implying it's the author's expectation. Including such estimates without empirical backing could mislead readers. If these tables were based on actual runs or references, that should be clarified. Otherwise, they're speculative and of limited value. Table 6.1 and 6.2 present real data (compile times for 100 runs of Hello World and 10 runs of a Terraform program) [38], which are useful to show compile overhead. They are clearly labeled and easy to read.
- **Labeling and Referencing:** All figures and tables seem to be properly numbered and titled, and the text references them appropriately. I did not encounter missing labels or confusion in referencing. For instance, the text frequently says "Şekil 5.9'da..." or "Çizelge 6.1'de..." which makes it easy to find the relevant figure/table. The captions are generally descriptive. Some captions, however, are very long (especially in Chapter 5) – essentially giving a mini-lesson within the caption. While thorough, a caption should ideally be a succinct description, deferring detailed explanation to the main text. Long captions can tire the reader. An example is the caption of Figure 5.8 (as seen in the PDF text, it spanned several lines with complex phrasing). It could perhaps be broken into multiple sentences or simplified.
- **Visual Quality:** Since I only saw the text extraction of figures, I cannot fully judge graphical quality. But the diagrams mentioned (like the Venn diagram in Figure 5.5 showing candidate classifications [39], or the flowcharts for the algorithm) suggest the author invested effort in visuals. If anything, a few figures try to convey too much at once (the layered search branching diagram Figure 5.13 might be quite busy). Still, using diagrams to illustrate the layered architecture, the algorithm flow, etc., is a strong point of the thesis. I suspect some figures (like the AST diagrams) might be small or complex, but since they are included, one hopes they were rendered in readable resolution. In conclusion, **figures and tables are generally useful and well-labeled**, but a few are either too detailed (appendix listing) or too complex to digest easily. Simplifying or summarizing those would improve the thesis.

Referencing and Citation

- **Literature Coverage:** The thesis excels in surveying relevant literature. It references classical works (Koza's foundational books from 1992 [40], Koza 2010 on human-competitive GP [41]), as well as recent research up to 2023. The bibliography includes key topics: Gene Expression Programming (Ferreira), strongly-typed GP (Montana), evolutionary program repair (Weimer 2009; Le Goues et al. 2012a & 2012b) [14] [42], and even very recent efforts like Higher Order GP in Haskell (Fernandes et al. 2023) [43]. It's clear the author did their homework to understand prior attempts at program synthesis in various languages (C, Java, Haskell, etc. are all cited). The thesis also references the Program Synthesis Benchmark Suite (PSB2) [44] and other contemporary resources, which shows the

literature review is up-to-date. I did not identify glaring omissions in the related work – it covers evolutionary computation basics, program synthesis, GP improvements (like bloat control, semantically driven GP, etc.), and even touches on relevant concepts from machine learning (like NEAT for neuroevolution, as an analogy). This comprehensive approach grounds the thesis well.

- **Use of Citations:** In-text citations use an author-year style (with "vd." for multiple authors in Turkish, which is equivalent to "et al."). This is consistent throughout. The author properly cites sources when making factual statements about others' work. For example, when stating *"GenProg was introduced by Le Goues et al. (2012a) to repair C code using tests"*, they cite the source [14]. I did not find instances of uncited claims about related work – credit is given where due. One minor issue is that a few broad claims in the thesis are not backed by citations. For instance, the statement about *"GP not being applied successfully to modern languages in 30+ years"* could have been supported by a reference or at least an example, but none is given (this is understandable as it's an introductory claim, though referencing Koza 2010 or a survey could strengthen it). Similarly, when discussing the rise of interest due to LLMs, the thesis doesn't cite any source; a citation to a relevant article or report on ML-driven coding could add credibility. These are relatively small oversights in an otherwise well-cited document.
- **Citation Quality and Formatting:** The reference list appears thorough and generally well-formatted, including details like authors, year, title, venue, etc. There are a few minor formatting inconsistencies: for example, I saw a reference to a Wikipedia page listed as "Anonymous 2023, Evolution Strategy, Wikipedia" [45]. Citing Wikipedia in a scholarly work is usually discouraged; it would be better to cite a textbook or seminal paper for something like Evolution Strategies. Also, some references show URLs and access dates (for code repositories or websites), which is good practice for non-traditional sources. The thesis even cites the GP field guide (Poli et al. 2008) and other online resources – these are appropriate and show a wide range of sources. The inclusion of very recent conference papers (GECCO 2021 for PSB2 [44], GECCO 2023 for HOTGP, etc.) is commendable – it means the author situated their work in the current context.
- **Completeness:** With around 80+ references, the thesis's bibliography is reasonably complete for a work of this scope. It balances older foundational works and newer innovations well. One could maybe suggest adding a citation on Large Language Models' code generation (to back the motivation), but given the focus is GP, this is not critical. Another tiny detail: the thesis references "Human-Competitive 2023" awards in passing [46], but I'm not sure if the reference list included a proper citation for that (it might; perhaps as a website). Ensuring every mention like that corresponds to a bibliography entry is important. Assuming the final reference list was checked, it likely has it. Overall, the citation practice in this thesis is solid. The author clearly identifies sources of algorithms and ideas, uses primary literature, and keeps citations up to date. Aside from the reliance on a few web sources (Wikipedia) instead of peer-reviewed alternatives, the referencing is professional. In a future revision, I'd suggest replacing informal sources with academic ones where possible, but in general there is no major issue – the thesis stands on a strong foundation of existing knowledge, properly cited.

---

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 849591.pdf

file://file_000000006c3c71f4a4adcc7d4e89cea4