Comparison of Fast Downward and ENHSP

Manuel Prida Sánchez

Universidad Internacional de La Rioja, Logroño, España

Abstract

Automated planning, as a core area of Artificial Intelligence, demonstrates its versatility by influencing a variety of domains, from robotic logistics to autonomous space exploration. In this discipline, heuristic planners are an essential component: algorithms that combine state space exploration with estimation functions to guide the search toward efficient solutions. Careful selection of the planner, search algorithm, and heuristic is critical to the performance, scalability, and quality of solutions obtained for complex problems.

This study evaluated the performance of FD (Fast Downward) and ENHSP (Enhanced Heuristic Search Planner) planners in the city car domain, analyzing their behavior in 20 satisfaction planning instances and 20 optimality planning instances. Fast Downward was confirmed to offer superior efficiency in both scenarios. In satisfaction planning, FD solved most instances faster and with generally shorter plans. In optimal planning, ENHSP is equally capable of generating optimal plans (using Weighted A* with weight 1 for the heuristic hmax()), but Fast Downward managed to solve more problems and did so in significantly less time. ENHSP showed significant scalability limitations on more complex problems in both modes, incurring more timeouts where FD managed to solve the instances or completed them more efficiently.

Key Words

Automatic planning, Fast Downward, ENHSP, search algorithms, heuristics.

I. INTRODUCTION

Automated planning, a fundamental pillar of Artificial Intelligence, has an impact on diverse areas such as robotic logistics [1] and autonomous space exploration [9]. At the heart of this discipline, heuristic planners are algorithms that combine state space exploration with estimation functions, known as heuristics, to guide the search toward efficient solutions [2]. Careful choice of planner, search algorithm, and heuristic is crucial for performance, scalability, and solution quality.

This study analyzes the performance of two heuristic planners: Fast Downward (FD) [5, 7], widely recognized in the community, and ENHSP (Enhanced Heuristic Search Planner) [4, 8], a less established but relevant tool for certain research, evaluating them in the city car domain [10].

The city car domain represents a simulated urban environment, structured as a grid where intersections are connected by road segments. The main challenge lies in dynamically managing this road network and coordinating the movement of several vehicles through it.

The state of the system is described by the current configuration of the roads (i.e., which segments are built and connect intersections), the availability of intersections (whether they are free), and the exact position of each vehicle, whether at an intersection or in transit on a segment. The initial state indicates how the grid is distributed at the beginning: which roads already exist and where the vehicles are located, usually in garages.

The main goal is for each car to reach its final destination, which is achieved when it reaches the intersection that has been assigned as its target (see Figure 1).

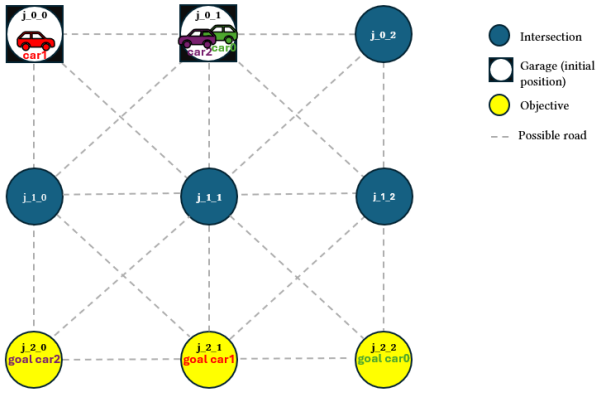


Fig. 1. Diagram of initial situation and objectives in instance-2 for the city-car-sequential-satisficing domain [10]. Source: Own elaboration.

The actions that the planner can perform include both infrastructure management and control of vehicle movement. New one-way roads can be built between two intersections, whether straight or diagonal, provided that the map geometry allows it and the final intersection is free. It is also possible to destroy an existing segment; a peculiarity of this action is that if there is a car on the destroyed section, it automatically returns to its intersection of origin.

With regard to vehicle movement, the model considers the entire route: from when they leave the garage to an intersection, passing through their entry onto a section of road and their exit to another intersection (provided it is free). Finally, there is a specific action to declare that the car has reached its destination and thus free up its position in the network. Each of these actions includes strict preconditions for execution, such as verifying that the section to be

traveled exists or that there are no blockages at the destination intersection.

The need to coordinate several vehicles simultaneously introduces an additional level of complexity: they must manage routes that avoid collisions and blockages, which means planning their movements intelligently and adapting dynamically to the conditions of the shared environment.

A key aspect of the study is the differentiation between the two approaches used to evaluate planners: satisfaction-oriented and optimality-oriented. Although both start from the same basic structure of the city car domain and use a cumulative cost function, the way in which this function is interpreted varies greatly depending on the scenario.

In the case of the satisfaction-oriented domain, called city-car-sequential-satisficing [10], the goal is simply to find a plan that works, without worrying about its cost. Although the actions define an increase in cost in their PDDL encoding, this is not considered when searching for the solution. The focus is on quickly finding a plan that meets the objective, making it a good testing ground for speed- and completeness-focused algorithms, such as Greedy Best-First Search, in combination with non-admissible heuristics such as add() or hadd().

In contrast, the city-car-sequential-optimal domain [10] introduces a more demanding approach: it is not enough to achieve the objective, but the total cost of the plan must be minimized. In this case, each action has an economic impact: building a straight road adds 20 units to the total cost, a diagonal adds 30, moving a car costs 1 per segment, and removing a segment involves a cost of 10 units. This approach forces the planner to perform a much more careful exploration of the solution space, seeking the most cost-efficient combination of actions.

To tackle this task, admissible heuristics such as hmax(), capable of reliably estimating the remaining cost, and optimal search algorithms such as A* or its weighted variant WA* with weight 1.0, which guarantee the quality of the result, are needed.

The sets of problems used to evaluate both scenarios were designed to gradually increase in difficulty. Each consists of 20 instances [10], adjusted to the requirements of the corresponding scenario (satisfaction or optimality). The order of complexity follows the criteria established in IPC 2014 for its deterministic segment [11], where instances are ordered according to how many planners were able to solve them. Thus, the simplest problems (such as instance-1) were solved by most planners, while the most complex ones (such as instance-20) were only solved by a few. This escalation in difficulty allows us to accurately measure how each planner behaves when faced with progressively more difficult challenges.

This report seeks to provide an empirical and well-founded view of the strengths and limitations of Fast Downward and ENHSP under different configurations. The comparison is carried out using relevant metrics such as the number of nodes generated, the time required to find a solution, and the length of the plans found.

It should be noted that all experiments were run in a local environment under homogeneous and controlled conditions: a 3.6 GHz processor, 8 GB of RAM, and a limit of 10 minutes per instance, ensuring fairness in the evaluation of both planners.

II. STATE OF THE ART

The field of automated planning has been a central area of research in artificial intelligence since its inception, with pioneering work laying the foundations for understanding how systems can reason about actions and their effects to achieve goals [6]. This field, which initially focused on logic and symbolic inference, has evolved significantly to address increasingly complex real-world problems. The view of planning as heuristic search has been fundamental to this evolution, allowing algorithms to explore complex state spaces efficiently [2]. This non-trivial approach combines the systematic exploration of the state space, which can be immensely large and often intractable by exhaustive search, with the intelligent use of heuristic functions. These heuristic functions are the main driver of efficiency, as they provide informed estimates of the “nearness” or estimated remaining cost to reach a target state from any intermediate state, thus guiding the search deliberately and effectively toward viable solutions. The sophistication of these heuristics is often the determining factor in a planner's performance.

Within this constantly evolving landscape, various planners have emerged that implement and refine these ideas, actively contributing to the advancement and practical application of the discipline. Fast Downward (FD) [5, 7] has established itself as one of the most influential and widely used planners in the automatic planning community. Its success is no accident; it lies in its remarkable flexibility and modular design, which allow it to integrate a wide range of search algorithms (from informed searches such as A* and Best-First Search to more complex approaches) and various high-performance heuristics. In addition, FD stands out for its robust ability to handle and exploit different features expressed in PDDL (Planning Domain Definition Language), the standard language for defining planning problems. One of its key innovations is the efficient transformation of PDDL problems into a multi-valued planning task representation. This internal representation allows it to apply a set of heuristic search techniques in an exceptionally robust and effectively scalable manner, achieving competitive performance in a large number of domains and in international planning competitions.

On the other hand, ENHSP (Enhanced Heuristic Search Planner) [4, 8] represents another important line of development in the field of forward search planners, distinguished by its particular focus on domains with numerical characteristics and continuous processes. This planner was specifically designed to address the challenges that arise when planning problems include numerical variables (such as fuel or inventory levels) and dynamics that change continuously over time. ENHSP focuses on the incremental construction of a search graph, where each step of the exploration is carefully guided by heuristics that are designed to effectively take into account and manage the complexities inherent in numerical dynamics and continuous states. Its specialized architecture allows it to be particularly competitive and effective in domains where precise resource management, optimization of numerical values, interaction with temporal or continuous states, and consideration of non-Boolean effects are prominent and critical components for problem solving.

Heuristics undoubtedly play a crucial and irreplaceable role in the efficiency of planners, as their quality and properties directly impact the ability to find solutions quickly and, in specific scenarios, with guaranteed optimality. Heuristics such as hmax() and hadd()/add() [3] are paradigmatic examples, widely studied and applied due to their distinctive properties and complementary behavior. hmax() is particularly valued for being an admissible heuristic; this

fundamental property means that it never overestimates the actual minimum cost of reaching the goal from any given state. This feature makes it ideal for use with optimal search algorithms, such as A*, which require guarantees of optimality in the solutions they find. The admissibility of hmax() ensures that if A* finds a solution, it will indeed be the lowest-cost one. In contrast, the add() heuristic (or its variant hadd()) is often more informative in practice and therefore heuristically stronger, providing tighter and more aggressive cost estimates. Although add() is inadmissible (meaning that it may, in some cases.

III. RESULTS

A. Satisfaction Comparison

The evaluation of the performance of Fast Downward (FD) and ENHSP planners in the field of satisfiable planning was designed to determine their ability to find any valid plan efficiently, prioritizing speed over cost optimality. To this end, Fast Downward was configured with the eager Greedy Best-First Search (GBFS) strategy and the add() heuristic. ENHSP used a similar configuration, also employing GBFS with the hadd() heuristic (see Table 1). Both heuristics are widely recognized for their effectiveness in guiding greedy searches toward fast solutions, despite being inadmissible [2]. This characteristic implies that they do not guarantee the optimality of the plan found, focusing instead on the speed of the first solution.

The set of problems used for this comparison consists of 20 instances from the city car sequential satisficing domain [10], meticulously designed to progressively increase in difficulty, allowing for an evaluation of scalability and robustness in the face of complex problems. The nature of these instances is specifically adapted for the satisficing scenario (SAT), where the existence of a plan is the primary objective and not its minimum cost.

The tests were run under controlled conditions (3.6 GHz CPU, 8 GB RAM, 600 s per instance), ensuring a fair comparison between the two experimental approaches.

TABLE I. SATISFACTION COMPARISON BETWEEN FD [5, 7] AND ENHSP [4, 8]. WHERE *N* ARE THE GENERATED NODES, *L* IS THE PLAN LENGTH, AND *T* IS THE EXECUTION TIME. TIMEOUTS ARE DENOTED AS "–". SOURCE: ELABORACIÓN PROPIA.

	FD with *GBFS* (*add*)			**ENHSP with *GBFS* (h_{add})**		
	N	*L*	*T*	*N*	*L*	*T*
instance-1	3.155	20	0,28	3.490	28	1,51
instance-2	1.561	29	0,29	1.842	32	0.942
instance-3	5.793	31	0,44	4.926	31	3,94
instance-4	5.274	46	0,46	3.260	44	1,33
instance-5	7.263	44	0,55	4.133	35	1,72
instance-6	652.076	28	18,75	216.702	37	141,86
instance-7	925.480	27	24,18	19.288	27	13,82
instance-8	665.894	25	18,76	27.288	25	26,65
instance-9	2.222.309	37	76,17	9.340	32	17,25
instance-10	799.802	38	33,01	71.679	44	131,50
instance-11	36.015	36	2,51	19.118	37	20,98
instance-12	47.157	40	3,09	14.404	44	24,57
instance-13	1.071.165	56	58,65	11.303	47	22,31
instance-14	1.136.610	42	50,67	45.270	52	97,03
instance-15	13.746	54	1,5	12.862	55	20,92
instance-16	1.544.220	46	88,9	83.373	58	195,75
instance-17	657.756	62	43,04	26.897	68	43,18
instance-18	1.650.991	56	96,15	13.613	63	37,70
instance-19	22.614	73	2,37	21.006	74	57,02
instance-20	40.880	62	3,76	37.343	72	88,26

In less complex instances (from instance-1 to instance-5), Fast Downward demonstrated a significant time advantage. For example, in instance-1, FD found a solution in just 0.28 seconds, while ENHSP took 1.51 seconds. This trend continued, with Fast Downward consistently being faster in these early tests, suggesting greater efficiency in the initial exploration of the state space.

As the complexity of the problems increased, Fast Downward maintained its time advantage in most cases. For example, in instance-6, FD solved in 18.75 seconds, while ENHSP took 141.86 seconds. This disparity in time performance became more pronounced in intermediate and complex problems, where Fast Downward scaled more efficiently. However, notable exceptions were observed: in instance-9, ENHSP was much faster (17.25 seconds vs. 76.17 seconds for FD), and in instance-13, ENHSP completed the task in 22.31 seconds compared to 58.65 seconds for FD. These variations suggest that, although FD is generally faster, the ENHSP heuristic can be effective in certain problem structures (see Figure 2).

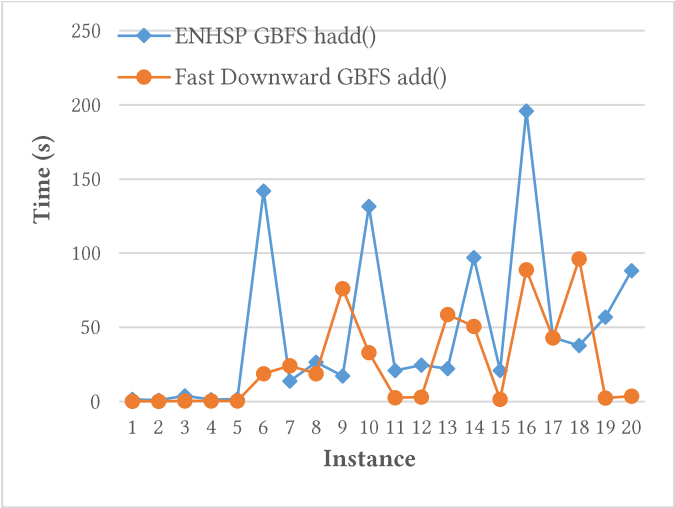


Fig. 2. Comparison of execution times for FD and ENHSP using 3.6 GHz and 8 GB RAM in the satisfaction domain. Source: Own elaboration.

The number of generated nodes provides insight into the computational effort of each planner during planning execution. In most satisfaction instances, Fast Downward (FD) generated substantially more nodes than ENHSP; for example, in instance-6, FD generated 652,076 nodes compared to ENHSP's 216,702. This suggests that, despite its time efficiency, FD explores a broader state space. Conversely, in instances where ENHSP outperformed FD in planning time (e.g., instance-9 and instance-13), it also generated far fewer nodes. Notably, in instance-9, ENHSP generated only 9,340 nodes versus FD's 2,222,309, indicating that the GBFS algorithm combined with the hadd() heuristic was exceptionally effective in guiding the search toward the solution more directly in those particular instances, drastically reducing the space explored (see Figure 3).

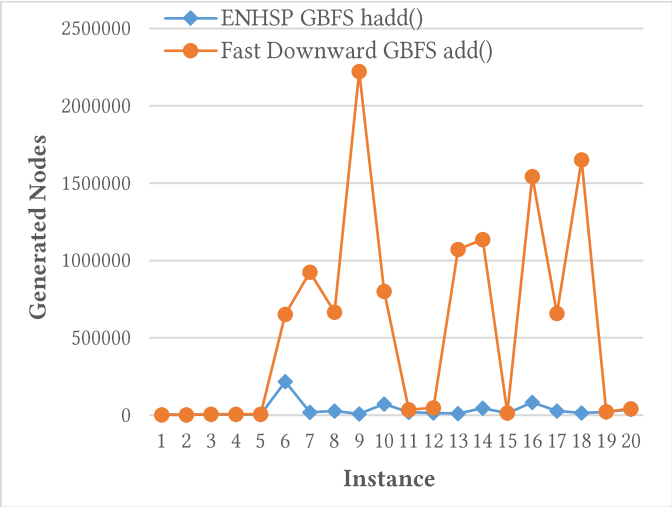


Fig. 3. Comparison of nodes generated with FD and ENHSP in the satisfaction domain. Source: Own elaboration.

With regard to plan length, no pattern of absolute dominance by any of the planners was observed, which is to be expected in satisfactory planning. In less complex instances (instances 1 through 5), Fast Downward generally produced shorter plans. For example, the plan for instance 1 was 20 steps with FD, compared to 28 with ENHSP. This suggests that, at the outset, the Fast Downward strategy is not only faster but also finds more direct paths.

However, as the complexity of the instances increased, ENHSP's ability to find shorter plans in certain situations became apparent. In instance-11, ENHSP generated a shorter plan (6 steps) than FD (8 steps). This variability underscores that, while satisfiable planning seeks any valid plan, the quality of the plan can vary considerably between planners that employ heuristics with the same basis but, due to the particularities of their implementation or how they interact with the search algorithm and the specific characteristics of each instance, explore the state space differently. Factors such as the order of node expansion, internal state management, or the way heuristic ties are resolved can lead to plans with different numbers of steps, even when the main objective is not minimization (see Figure 4).

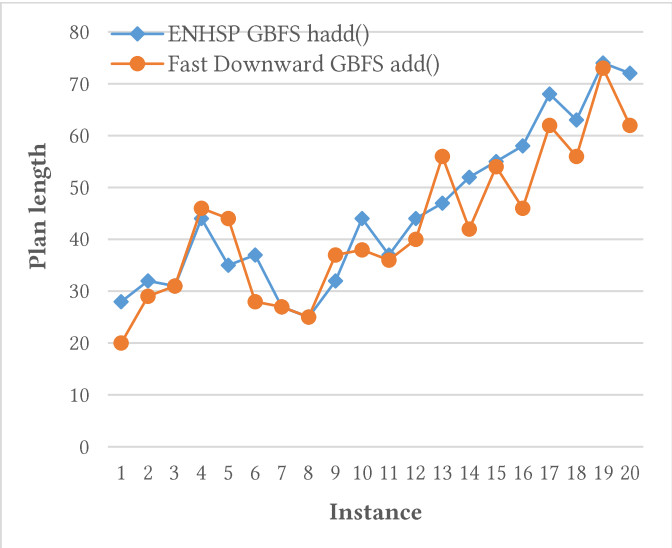


Fig. 4. Comparison of plan length for FD and ENHSP in the area of satisfaction. Source: Own elaboration.

In summary, the results in satisfactory planning confirm that Fast Downward is the most efficient and robust planner for the city car domain. It solves faster, with lower computational cost (fewer nodes generated) and produces shorter plans in most cases, making it a solid tool for contexts where response time is critical. ENHSP, while functional, shows scalability limitations that result in timeouts in complex problems.

As an example and in order to understand one of the plans generated in the satisfaction domain, the plan generated for instance-2 was analyzed and represented using diagrams (see Figure 1 at the beginning of the study and Appendices 1, 2, and 3). Annex 3 shows how a road is unnecessarily destroyed, confirming that the plan, although only involving a single movement (destroy_road junction2-1 junction2-2 road1), is not optimal.

B. Optimality Comparison

In optimality-oriented planning, the goal is to find the plan with the lowest possible total cost. In the city-car-sequential-optimal domain [10], this cost is defined by the total-cost function, where actions such as building straight or diagonal roads, moving cars, or destroying roads have specific costs (20, 30, 1, and 10 units, respectively). Achieving optimal solutions requires informed algorithms and admissible heuristics, such as hmax(), which allow A* to guarantee optimality.

In this evaluation, Fast Downward used A* with hmax(), and ENHSP used Weighted A* with weight 1 (equivalent to pure A*) and the same heuristic (see Table 2). Both configurations were run under equal conditions (8 GB of RAM, 600 seconds per instance) on 20 city car domain problems, designed with incremental difficulty to evaluate their performance under optimality requirements.

This approach allows us to compare not only the effectiveness but also the temporal efficiency of both planners when faced with maximum quality solutions in demanding contexts.

TABLE II. COMPARISON OF OPTIMALITY BETWEEN FD [5, 7] AND ENHSP [4, 8]. WHERE N ARE THE GENERATED NODES, $L(C)$ IS THE PLAN LENGTH AND ITS COST, AND T IS THE EXECUTION TIME. TIMEOUTS ARE DENOTED AS "–". SOURCE: OWN ELABORATION.

	FD with A* (h_{max})			**ENHSP with A*** (h_{max})		
	N	*L (C)*	*T*	*N*	*L (C)*	*T*
instance-1	10.607	12 (46)	0,18	9.619	12 (46)	0,74
instance-2	65.978	10 (64)	0,32	40.914	10 (64)	3,19
instance-3	58.991	18 (50)	0,4	58.276	18 (50)	3,19
instance-4	46.901	16 (48)	0,38	43.361	16 (48)	2,60
instance-5	58.991	18 (50)	0,39	58.276	18 (50)	3,22
instance-6	253.717	22 (52)	1,37	223.350	22 (52)	13,75
instance-7	253.717	22 (52)	1,37	223.350	22 (52)	13,31
instance-8	10.034.358	29 (76)	56,45	-	-	-
instance-9	6.790.983	32 (58)	41,24	5.941.355	32 (58)	511,34
instance-10	6.790.983	32 (58)	41,16	5.936.749	32 (58)	517,99
instance-11	388.667	15 (68)	2,09	251.734	15 (68)	28,65
instance-12	2.283.046	17 (78)	13,62	1.402.686	17 (78)	229,08
instance-13	7.914.942	23 (82)	64,43	-	-	-
instance-14	-	-	-	-	-	-
instance-15	4.340.691	17 (70)	49,51	-	-	-
instance-16	-	-	-	-	-	-
instance-17	130.661	14 (48)	1,86	146.927	14 (48)	26,23
instance-18	4.049.094	17 (70)	45,28	-	-	-
instance-19	-	-	-	-	-	-
instance-20	-	-	-	-	-	-

Fast Downward demonstrated clear superiority in planning time for optimal problems. In low-complexity instances, such as instance-1, FD solved in 0.18 seconds, while ENHSP took 0.74 seconds. In instance-2, the differences begin to be greater, with 0.32 seconds for FD and 3.19 seconds for ENHSP.

As complexity increases, the performance gap becomes more pronounced. In instance-6, FD solved in 1.37 seconds, while ENHSP took more than 13 seconds (13.75 s), a difference of 10 times. For instance-4, FD also took 10 times less time than ENHSP to provide an optimal solution.

On the most challenging problems, Fast Downward maintained its efficiency, while ENHSP showed clear limitations. In instance-9 and instance-10, FD solved in just over 41 seconds, while ENHSP took more than 510 seconds in both cases (almost 9 minutes), being more than 12 times slower. Similarly, in instance-11, FD completed in 2.09 seconds, while ENHSP required 28.65 seconds.

ENHSP experienced timeouts in 8 of the 20 instances tested, representing 40% of the test suite, indicating a significant limitation in its ability to scale to complex scenarios. In contrast, Fast Downward solved more instances and did so more quickly. For example, FD solved instance-17 in 1.86 seconds and instance-18 in 45.28 seconds, problems that ENHSP was unable to complete. Fast Downward also suffered timeouts in instances 14, 16, 19, and 20 (20% of the test suite).

It should be noted that FD showed a more progressive and predictable evolution over time as the difficulty of the instances increased, while ENHSP shows abrupt jumps in its response times. This lack of linearity in ENHSP's behavior compromises its reliability in contexts where a reasonable estimate of planning time is required (see Figure 5).

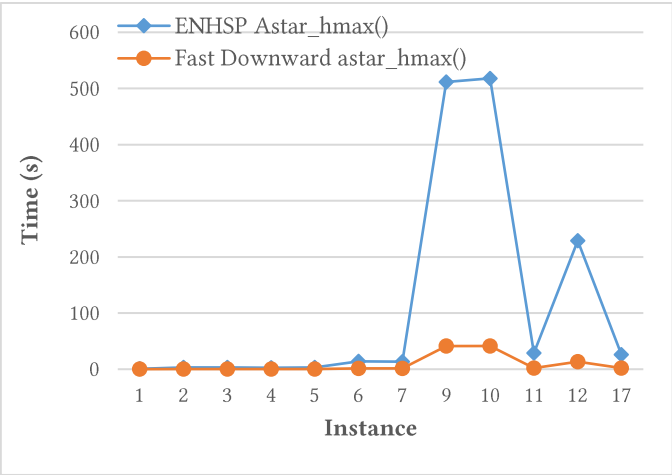


Fig. 5. Comparison of execution times for FD and ENHSP using 3.6 GHz and 8 GB RAM in the optimality domain. Note: Only instances solved by both planners have been considered in preparing this graph. Source: Own elaboration.

Regarding the number of nodes generated (N), which indicates how much of the search space is explored, a clear trend can be seen: in most cases where both planners managed to find a solution (instances 1 to 7, 9 to 12, and 17), ENHSP proved to be more efficient, as it generated fewer nodes compared to FD. This points to a more effective use of heuristics or more aggressive pruning techniques by ENHSP in those specific instances. However, this advantage has its limitations. Fast Downward showed greater robustness in the face of complex problems, as seen in instances 8, 13, 15, and 18, where it did manage to plan a solution while ENHSP did not (indicated with a "-"). This behavior suggests that FD handles large or more complicated search spaces better, maintaining its ability to find solutions even when ENHSP does not progress. Instances 14, 16, 19, and 20, on the other hand, remained unsolved by both planners (see Figure 6).

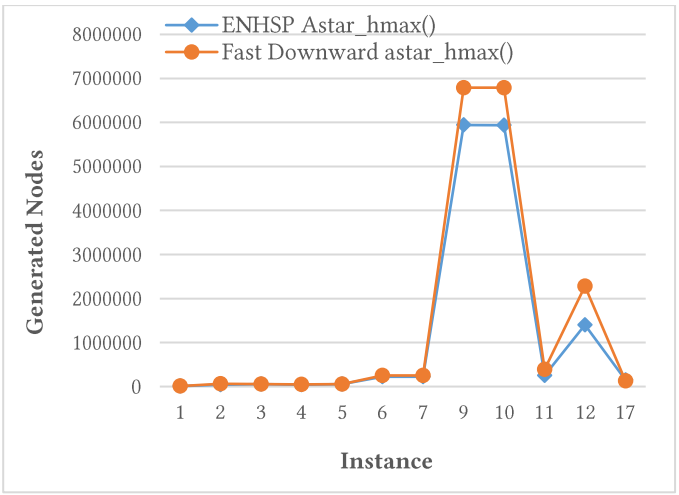


Fig. 6. Comparison of nodes generated with FD and ENHSP in the optimality domain. Note: Only instances solved by both planners have been considered in preparing this graph. Source: Own elaboration.

In terms of plan length (L) and, more importantly, cost (C), the results show complete agreement in all instances solved by both methods. In other words, the plans generated have the same cost and number of actions, confirming that both systems, working with A* and an admissible heuristic such as hmax(), are identifying equivalent optimal paths. This equality reinforces the validity of their results in terms of optimality.

In short, it can be concluded that both ENHSP and Fast Downward are capable of finding optimal solutions from a cost perspective. However, they differ in their behavior: ENHSP stands out for its greater efficiency when solving the problem, while FD offers broader coverage by being able to find solutions even in complex cases, despite generating more nodes in certain scenarios.

IV. CONCLUSIONS

This analysis has revealed notable and consistent differences between Fast Downward and ENHSP planners within the city car domain [10], both in satisfactory and optimal planning environments. Based on these observations, firm conclusions can be drawn about the efficiency, scalability, and quality of the plans generated by each approach.

In the satisfactory scenario, where the main objective is to quickly obtain any valid plan regardless of its cost, the results show a clear advantage for Fast Downward. Configured with the eager Greedy Best-First Search (GBFS) strategy and the add() heuristic, FD showed consistently superior performance. It successfully solved all 20 instances in the test suite, while ENHSP, using GBFS with the hadd()

heuristic, presented more mixed results. Although ENHSP achieved slightly lower times in some simple instances, Fast Downward was significantly faster in most cases of intermediate and high difficulty. An illustrative example can be seen in instance-6, where FD obtained a solution in 18.75 seconds compared to the 141.86 seconds required by ENHSP. In addition, Fast Downward showed greater efficiency in exploring the search space, generating considerably fewer nodes in most tests, which also had a positive impact on the length of the plans produced. Although in some specific instances (such as instance-9 or instance-13) ENHSP achieved shorter routes, the overall consistency of FD was unquestionable, satisfactorily meeting the objectives in the 20 instances evaluated.

For its part, ENHSP, configured analogously with GBFS and hadd(), did manage to complete all instances of the scenario satisfactorily, but exhibited significant deficiencies in terms of comparative efficiency. When faced with more complex problems or more intensive coordination requirements, planning time increased considerably. This loss of performance was particularly evident in instances such as instance-6, instance-10, and instance-16, where its response times were substantially higher than those of Fast Downward.

In the optimal planning scenario, which requires finding the lowest possible cost plan, Fast Downward, using A* together with the hmax() heuristic, once again came out on top. Both planners were configured with admissible heuristics, so when they managed to solve an instance, they generated optimal plans equivalent in length and cost. However, the differences in the time required to achieve this were notable. Fast Downward completed 16 of the 20 instances in times clearly inferior to those obtained by ENHSP. A representative case is instance-9, in which FD found the plan in 41.24 seconds, while ENHSP needed 511.34 seconds. The scalability of ENHSP was severely compromised: it failed to finish within the time limit in 8 of the 20 tests, compared to only 4 timeouts by Fast Downward. This indicates that, despite its theoretical ability to achieve optimality, the computational cost per node in ENHSP is significantly higher, which seriously limits its applicability in environments with time constraints or large-scale problems.

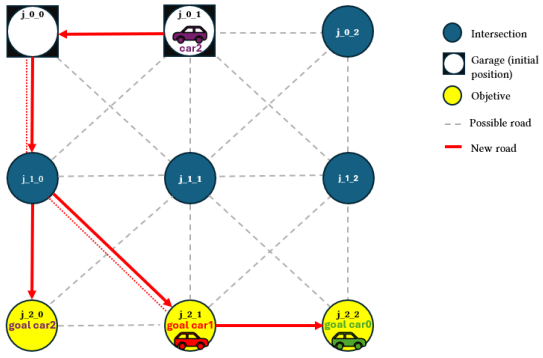
Overall, the data obtained throughout this research clearly shows that Fast Downward is a more effective, scalable, and robust option for both satisfactory and optimal planning within the analyzed domain. Its outstanding performance in the satisfactory scenario, with fast resolution times and generally shorter plans, makes it a valuable tool for real-time applications. In turn, its ability to produce optimal solutions remarkably quickly and to solve a greater number of complex instances positions it as a reliable choice in contexts where plan quality is a priority. This superiority is not only due to the search strategies used, but also to the optimized implementation of FD, which allows the complexity of the search space to be managed efficiently, reaffirming its role as a benchmark in the field of classical heuristic planning.

ENHSP, on the other hand, is a conceptually sound and functionally valid tool, particularly valued for its ability to find optimal plans when properly configured. However, it suffers from obvious limitations in terms of practical efficiency, particularly in domains with large size or high density of connections between elements. This gives it a relevant role mainly in the field of research, where its numerical capabilities, expressiveness, and heuristic flexibility can be decisive, and where runtime requirements are not as critical as in production environments.

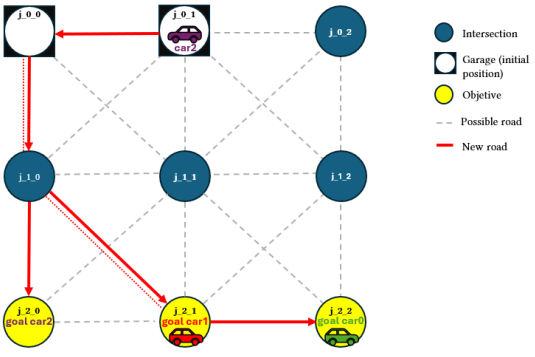
Looking ahead to future lines of work, it would be pertinent to explore possible internal improvements in the ENHSP architecture that would reduce the computational cost per node, such as optimizations in data structures, improvements in the grounding process, or in search boundary management. It is also proposed to extend the comparison of planners to mixed or hybrid domains, including numerical or temporal constraints, where the particular strengths of ENHSP could be more advantageous. Another line of interest would be to increase the set of instances evaluated and to perform detailed monitoring of critical errors or memory failures during execution, which would further enrich the comparative analysis and allow the evaluation criteria to be adjusted according to the requirements of the domain.

In short, this study presents a rigorous comparison between Fast Downward, one of the most established and widely used planners in the field of automatic planning, and ENHSP, a research tool that, although less widespread and with less bibliographic support, offers interesting functionalities for specific scenarios. The results clearly identify the strengths and limitations of each approach, providing a valuable basis for both future research and practical applications that require efficient and robust solutions.

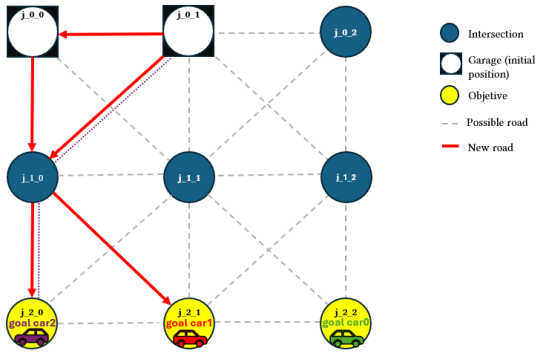
APPENDIX



Appendix 1. Instance-2 resolution diagram for the city-car-sequential-satisficing domain [10] – adjustment of roads and car movements. Source: Own elaboration.



Appendix 2. Instance-2 resolution diagram for the city-car-sequential-satisficing domain [10] – movements of car1. Source: Own elaboration.



Appendix 3. Instance-2 resolution diagram for the city-car-sequential-satisficing domain [10] – adjustment of roads and car movements. Source: Own elaboration.

BIBLIOGRAPHY

[1] Adeoye, Y., Onotole, E. F., Ogunyankinnu, T., Aipoh, G., Osunkanmibi, A. A., & Egbemhenghe, J. (2025). *The function of AI in dynamic route planning for transportation. World Journal of Advanced Research and Reviews, 25*(2). https://doi.org/10.30574/wjarr.2025.25.2.0214

[2] Bonet, B., & Geffner, H. (2001). *Planning as heuristic search.* Artificial Intelligence, 129(1-2), 5–33.

[3] Corrêa, C., Fišer, D., Geffner, A., Bonet, B., & Geffner, H. (2021). Delete-Relaxation Heuristics for Lifted Classical Planning. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).* https://ai.dmi.unibas.ch/papers/correa-et-al-icaps2021.pdf

[4] ENHSP. (n.d.). *ENHSP: Enhanced Heuristic Search Planner.* Recuperado de https://sites.google.com/view/enhsp/

[5] *Fast Downward* contributors. (2025, February 7). *The Fast Downward domain-independent classical planning system* (Version 24.06.1) [Source code]. GitHub. https://github.com/aibasel/downward

[6] Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence, 2*(3-4), 189-208.

[7] Helmert, M. (2006). *The Fast Downward Planning System.* Journal of Artificial Intelligence Research, 26, 191–246.

[8] hstairs. (2025). *enhsp* [Código fuente]. GitHub. https://github.com/hstairs/enhsp

[9] Pell, B., Gamble, E. B., Gat, E., Keesing, R., Kurien, J., Millar, W., Plaunt, C., & Williams, B. C. (1999). A Hybrid Procedural/Deductive Executive for Autonomous Spacecraft. *Autonomous Agents and Multi-Agent Systems, 2*(1), 7-22.

[10] Potassco. (n.d.). *pddl-instances/ipc-2014/domains.* GitHub. Recuperado el 11 de junio de 2025, de https://github.com/potassco/pddl-instances/tree/master/ipc-2014/domains

[11] Vallati, M., Chrpa, L., & McCluskey, T. L. (2018). *What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask).* The Knowledge Engineering Review, 33(e3), 1-36. doi:10.1017/S0269888918000012