Planning Complex Processes for Autonomous Vehicles by Means of Genetic Algorithms

Nermeen M. Ismail[1], Magda B. Fayek[2], Ashraf A. Wahab[1], and Nevin M. Darwish[2]

[1] Electronic Research Institute, Department of Computers and Systems , AI Group
[2] Cairo University, Faculty of Engineering, Department of Computer Engineering

Abstract. Autonomous robots are becoming an increasingly important tool for military, space exploration, and civilian applications. A key requirement for controlling mobile autonomous robots is the ability to express vehicle activity models as complex processes. This work presents PGen[1]; a generative activity planner that translates intended state evolution to an action plan. PGen supports generative planning with complex processes via three main features. First, PGen goal plans and activity models are encoded using Reactive Model-based Programming Language (RMPL). Second, PGen represents goal plans, plan operators and plan candidates with a uniform representation called Temporal Plan Networks (TPN). Finally, PGen uses Genetic Algorithms as a novel approach for TPN-based planning. PGen has been successfully implemented and tested, simulation results are very promising.

1 Reactive Model-based Programming Language

Model-based programming languages help model the relationships between various robot states by incorporating features such as concurrency, metric constraints and durations, functionally redundant choice, contingencies, and synchronization. RMPL is a rich language for describing activity models of autonomous reactive systems. Designed to help complexity, RMPL is object-oriented and supports high-level programming features such as abstraction, encapsulation, and inheritance. RMPL is a process algebra that enables programmers to easily encode arbitrarily complex activity models and mission control programs[4][11].

2 Temporal Plan Networks

RMPL allows a programmer to specify complex processes in terms of an easy representation that defines the evolution of state variables. To enable fast planning, RMPL programs are converted into equivalent graph structures called Temporal Plan Networks (TPN). TPN are useful in that they compactly encode the space of possible state evolutions expressed by an RMPL program. TPNs are collections of events and episodes representing processes that may have their own sub-goals. An episode can have a Primitive Activity (PA), a Non-Primitive

Activity (NPA), a state query (ASK) and a state assertion (TELL). A PA is a simple activity that is not composed of any further activities. An NPA is an activity that is composed of further events, episodes and PAs. An ASK is a request that a particular assignment is being achieved for some period of time, so it represents open conditions that need to be satisfied. A TELL is an assertion that a particular assignment is already achieved for some period of time. Once a program has been converted to a TPN, it can be processed using efficient network algorithms to perform search, scheduling, etc[7].

3 PGen within Kirk

The contribution of this paper is part of Kirk. Kirk is a mission-level model-based executive, designed to control mobile autonomous robots in rich environments, such as rovers exploring the surface of Mars or unmanned aerial vehicles flying for search and rescue missions. PGen refines the complex sequence of goals (mission plan) into an actionable activity plan using existing activity library. This function was previously provided by another planner called Spock[7]. Spock uses A* search algorithm to search for a complete and consistent solution plan. In this paper, we propose PGen that uses Genetic Algorithms as a novel approach for TPN-based planning. Genetic Algorithms have shown successful performance when used to generate action plans represented as TPNs. Figure 1 shows PGen planner as part of Kirk executive. The activity library database contains all pos-

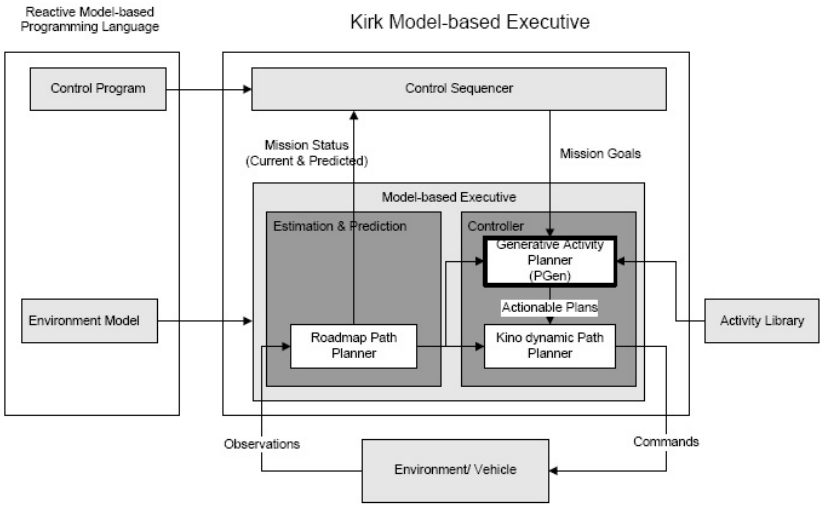

Fig. 1. PGen generative planner as part of Kirk model-based executive architecture

sible activities that the vehicle can perform. These activities are represented as

Temporal Plan Networks and are used by PGen. Figure 2 shows PGen generative planner control flow.

4 PGen Control Flow

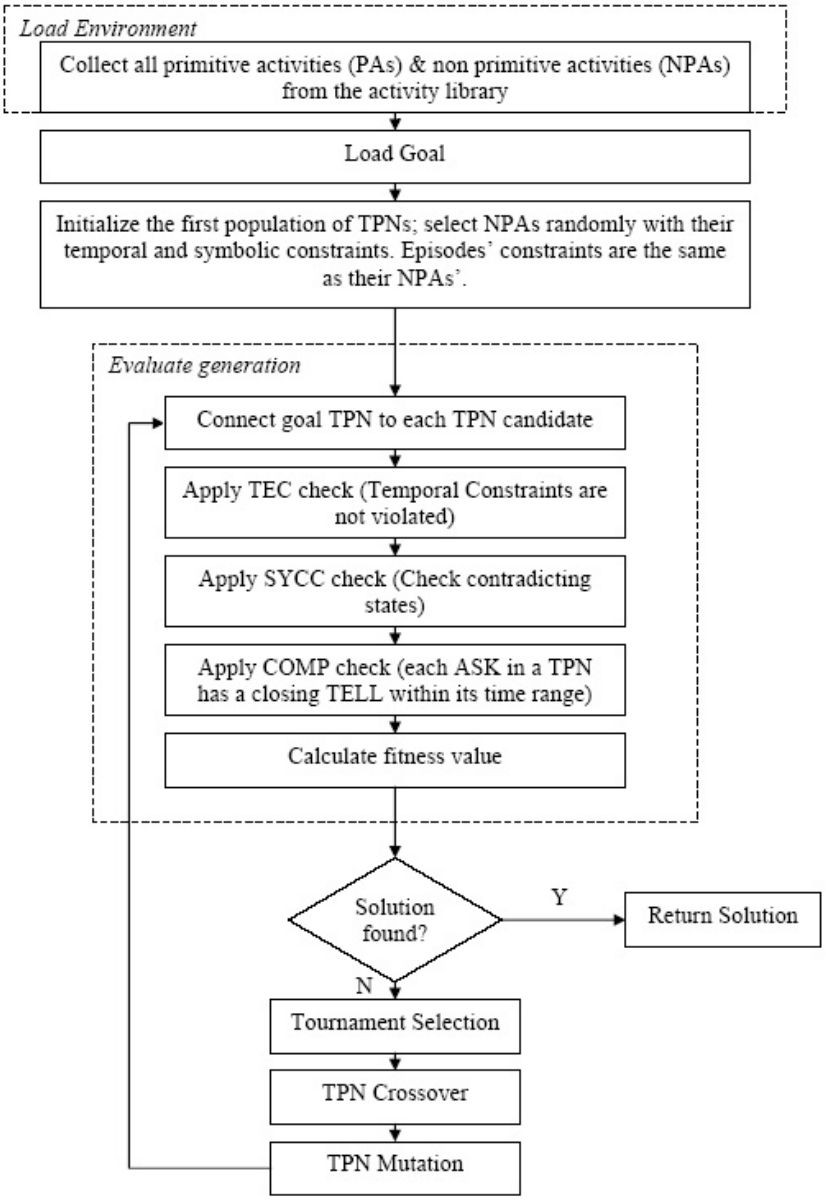

Fig. 2. PGen control flow

4.1 Initialization and Chromosome Structure

PGen starts its genetic loop by creating an initial population of chromosomes. Each chromosome represents a TPN structure that consists of events and episodes. Each episode has zero or more NPAs collected from the activity library. Each TPN has a start and an end event. Number of parallel paths and number of events per path are user-adjusted parameters however; the current implementation sets them randomly within specific ranges.

4.2 Fitness Function

Not all TPN candidates are executable on mission hardware. This is either because some open conditions (ASK) within the TPN are not satisfied, or some combinations of TPN constraints are conflicting. The resulting solution TPN is said to be executable if it is both consistent and complete. PGen evaluates each TPN candidate based on its temporal consistency (TEC), symbolic constraints consistency (SYCC) and completeness (COMP). TEC requires that a valid temporal assignment to each event exists such that no temporal constraints are violated. SYCC ensures that there are no overlapping intervals with conflicting constraints. COMP requires that all open questions represented by ASK constraints are satisfied. Fitness is calculated as follows: if a TPN candidate is consistent and complete, its fitness is zero. Otherwise, it takes a value based on the number of events and the number of open conditions (1).

$$F = \begin{cases} 0 & \text{Candidate passedTEC, SYCC and COMP} \\ EV + OpASKs & \text{Candidate passed TEC, SYCC but failed COMP} \\ MAXFITNESS & \text{Candidate failed TEC} \end{cases} \quad (1)$$

where EV=No of events and OpASKs= No of open ASKs.
Hence, a candidate's fitness is estimated over three phases as shown in Fig.3.

4.2.1 TEC In order to measure a candidate TEC, temporal constraints have to be reformulated into an equivalent graph called a distance graph[12]. A distance graph is a graphical encoding of each upper and lower bound in a graph. Once the distance graph for a given TPN has been constructed, one can easily determine temporal consistency by using a negative cycle detection algorithm, as the existence of a negative cycle implies that there is a set of temporal constraints that can not be satisfied. All Pairs Shortest-Path (APSP) algorithms has been used to detect negative cycles in the distance graph. PGen uses Johnson's Algorithm[6] to detect negative cycles for distance graphs.

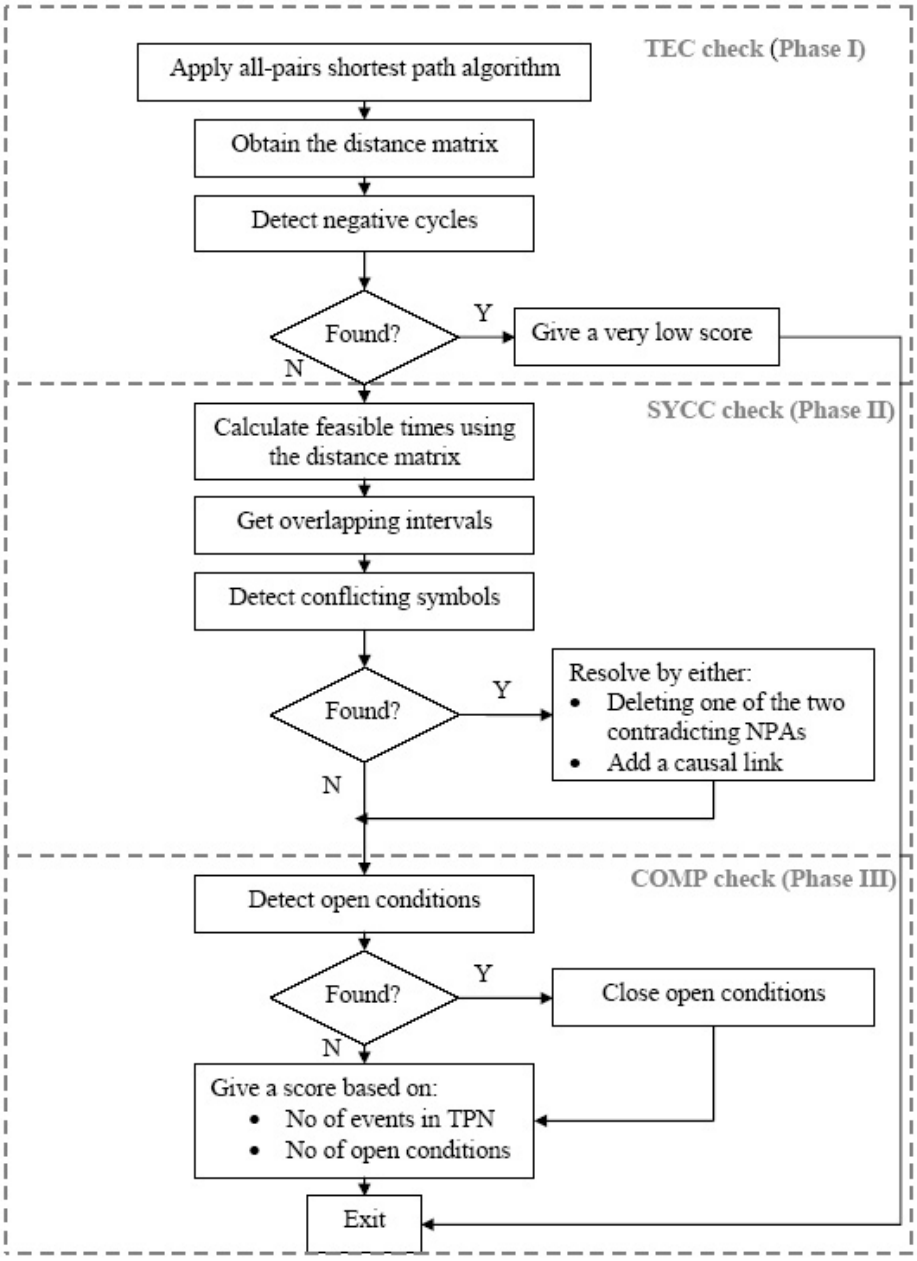

Fig. 3. Fitness calculation phases

4.2.2 SYCC An incompatibility exists when there are two episodes in TPN representing overlapping intervals of time, labeled with symbolic constraints that conflict. Two symbolic constraints conflict if one is either asserting or requesting that a condition is true, while the other is asserting or requesting that the same condition is false. For example TELL (Not C) and ASK (C) conflict, as do ASK(C) and ASK (Not C). Since such condition pairs can never both be satisfied at the same time, they represent one form of plan inconsistency.

Conflict Detection In order to detect incompatibilities, we must first compute the feasible time bounds for each temporal event, and then use these bounds to identify potentially overlapping intervals that are labeled with conflicting symbolic constraints. These bounds can be computed by solving an all-pairs shortest path problem over the distance graph representation. PGen uses results previously obtained from Johnson's Algorithm to solve all-pairs shortest path problem.

Conflict Resolution PGen resolves the symbolic conflict either by deleting one of the two contradicting activities or by constraining the time ranges of the start and end points of the intervals to ensure they will not overlap.

4.2.3 COMP An open condition is represented by an episode labeled with an ASK constraint. It represents the request for a condition to be satisfied over a certain interval of time. When PGen calculates the completeness of a TPN candidate, it first checks to see how many ASKs are satisfied by closing TELLs. It tries to satisfy or close these open conditions. Once an interval that may satisfy this open condition is found, a causal link can be added to force the interval to contain the interval of the open condition. A causal link is an episode with [0, +INF] time-bounds and no attached ASKs, TELLs, or any activities. They are mainly used to order plan activities and force a certain sequence of events to occur. Finally, PGen calculates a candidate's score according to the number of events in TPN and the number of unsatisfied ASKs.

4.3 Elitism

Elitism means that the best chromosome (or a few best chromosomes) is copied to the population in the next generation. The rest are chosen in classical way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date.

4.4 Tournament Selection

Based on earlier research results [3], PGen uses Tournament Selection rather than other selection strategies like Roulette Wheel Selection. Tournament Selection is one of many methods of selection in Genetic Algorithms which runs a "tournament" among a few individuals chosen at random from the population

and selects the winner (the one with the best fitness) for crossover. Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected and vice versa.

4.5 TPN Crossover

PGen uses two crossover operators to perform the evolution; namely:
1. TPN Multiple Points Crossover: This is a novel crossover operator especially designed for Temporal Plan Networks. It divides a TPN at some randomly chosen cut sets. A cut set consists of some episodes at which a TPN will be divided into two parts.
2. TPN Single Activity Swap Crossover: An episode is selected at random from each chromosome and contents are swapped. Please refer to the upcoming example in the next section.

4.6 TPN Mutation

PGen depends much on mutation operations to investigate the search space. It uses the following proposed mutation operators:
1. TPN Activity Addition Mutation: An NPA is selected from the activity library and inserted at a random episode with no NPA.
2. TPN Activity Deletion Mutation: An episode is selected at random in the TPN candidate and its NPA is removed.
3. TPN Internal Activity Swap Mutation: Two episodes are selected at random in the TPN candidate and contents are swapped.
4. TPN Activity Change Mutation: An episode is selected at random and its NPA is replaced by another one selected from the activity library.

5 Example

Consider that there is a ship sinking in the sea and there is one person on it. It is required to fetch that person then put him on the medical ship. So, the mission programmer writes the control program in Fig.4.

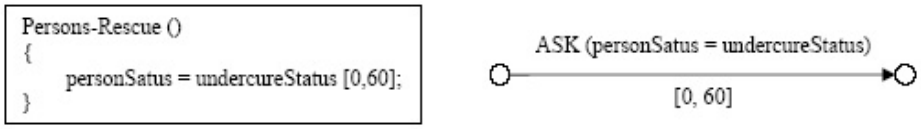

Fig. 4. Control program RMPL and its equivalent TPN for rescue mission

Assume that the activity library contains four activities: Search-For-Sunken-Object(), Determine-Object-Type(), Rescue-Sunken-Person() and Extinguish-Fire(), see Fig.5.

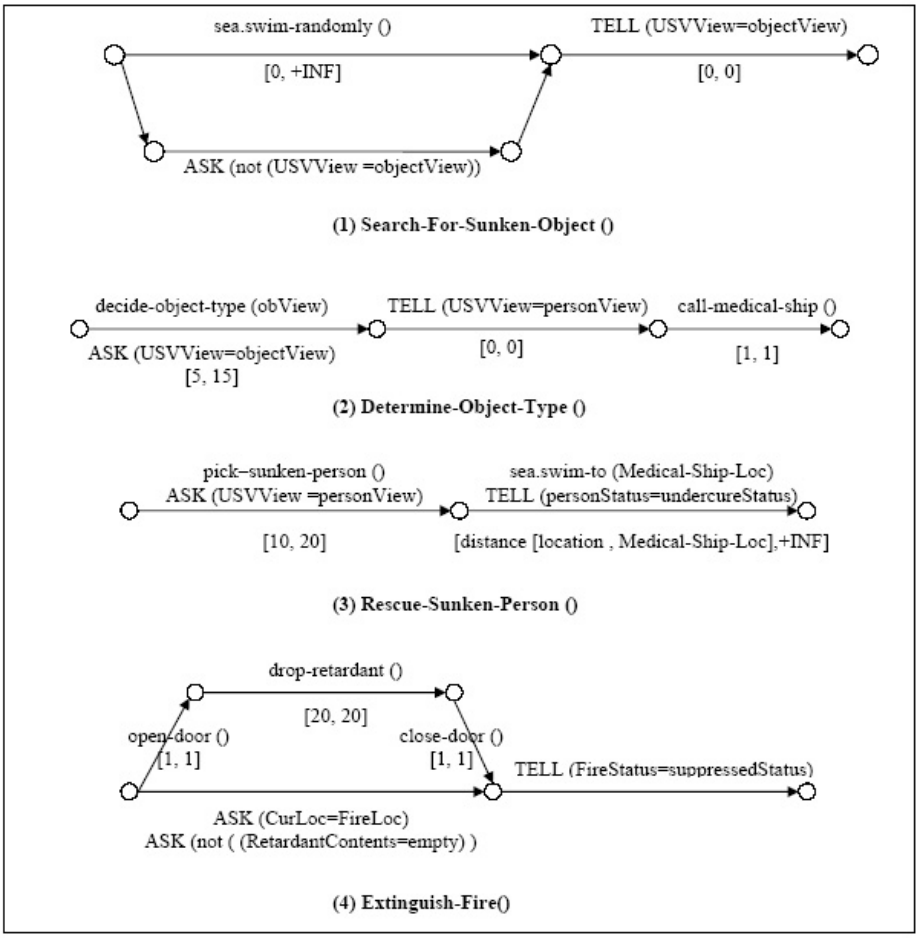

Fig. 5. Activity Library

5.1 Initialization

Assuming that PGen produces the initial generation (as explained before), given in Fig.6 an example TPN candidate.

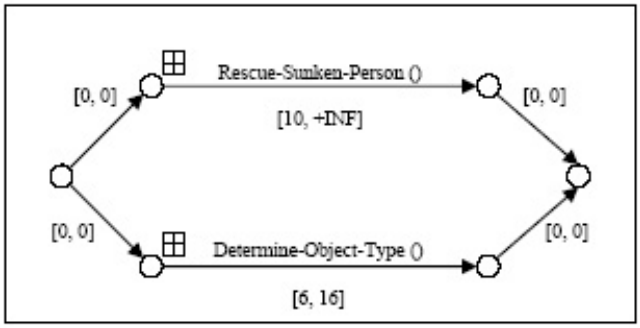

Fig. 6. A TPN candidate selected from the first generation

5.2 Fitness

For fitness calculation, PGen transforms Temporal Plan Networks into its corresponding distance graphs. Then it applies Johnson's Algorithm in order to obtain the distance matrix. Figure 7 shows the distance graph, distance matrix and the calculated feasible times for the TPN candidate at hand. It is clear that it does not contain any negative cycles, thus it passes TEC check. After

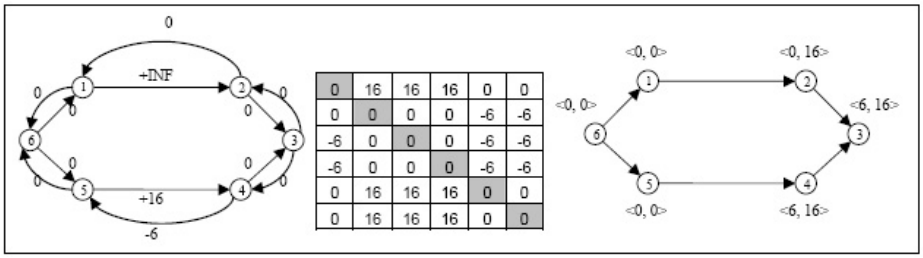

Fig. 7. Distance graph, Distance matrix and Calculated feasible times for TPN candidate in Fig.6

that, TPN candidates are expanded and go through SYCC check, see Fig.8. The candidate at hand does not contain any conflicting symbols, so it passes this check successfully. Finally, COMP check is conducted; PGen connects the goal to each TPN candidate, and counts the number of unsatisfied ASKs. Figure 8 shows the candidate at hand expanded with the goal connected to it. The open conditions are: ASK (USVView =personView), ASK (USVView=objectView) and ASK (personStatus=underCureStatus). The first condition is the only one satisfied. So, the fitness value= Number of events + Number of unsatisfied ASKs = 9+2 =11.

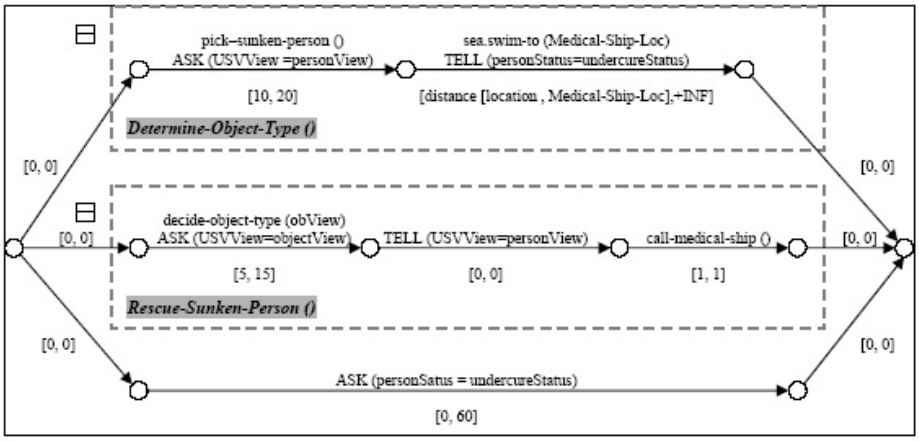

Fig. 8. TPN candidate in Fig.6 expanded and connected to goal

5.3 Crossover

Assume that the TPN candidate in Fig.6 is selected for crossover along with candidate in Fig.9.b. TPN Multiple Points Crossover is applied and crossover points are selected at random. Events with gray color in Fig.9.a and Fig.9.b are the crossover points. Two offsprings are formed in the next generation as shown in Fig.9.c and Fig.9.d.

5.4 Fitness

When Fitness is evaluated for the two offsprings, PGen finds that the child in Fig.9.c passes the three checks, hence it takes 0 fitness value and is returned as a solution, see Fig.10.

6 Experimental Results

PGen was implemented in C++ and tested on a Pentium IV 3 GHz processor with 1 GB of RAM running Windows XP SP2. It run on a series of test problems to check its effectiveness. Test data consists of 66 problems of different complexities. Complexity is measured as the number of Primitive Activities and Non-Primitive Activities required solving the problem. For correct parameter selection, the effect of changing Elitism Size, Tournament Size were studied. We will study the effect of these two parameters on the probability to reach a solution. 20 runs were performed for the same problem then we get number of successful runs that found a solution. Failed runs are those that did not reach a solution. In addition, performance analysis (amount of processing required to solve a problem)[2] has been conducted to specify suitable crossover rate[1].

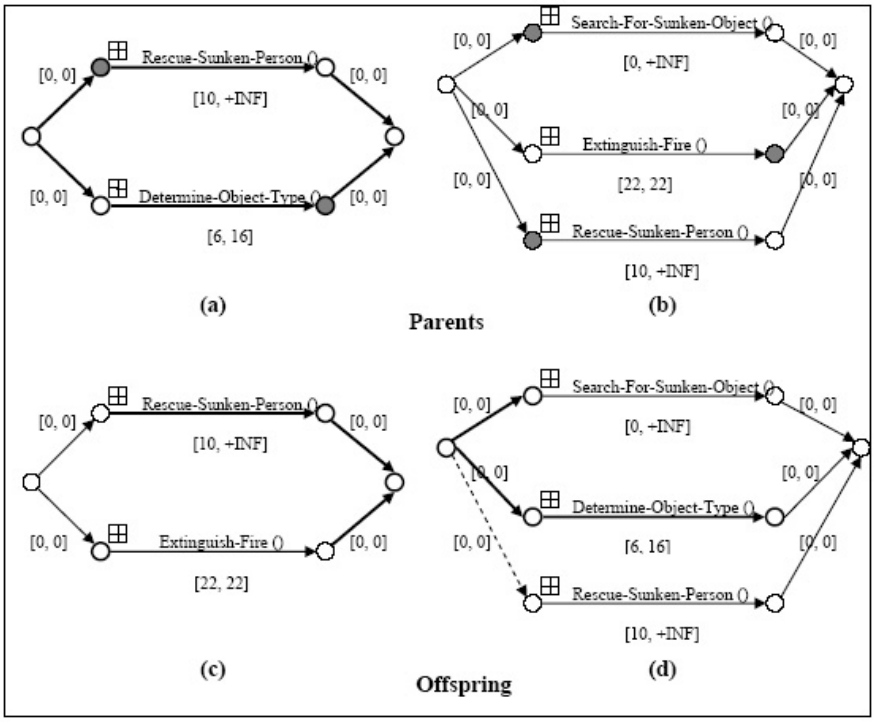

Fig. 9. TPN Crossover applied to parents and offsprings are formed

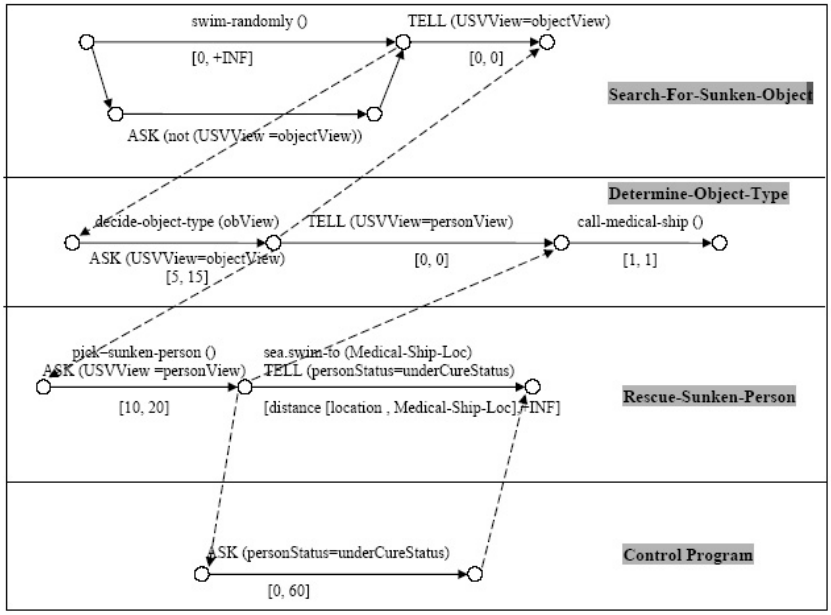

Fig. 10. A solution is found

6.1 Elitism Size

Figure.11 shows the Elitism Size against probability to reach a solution for three problems. Note that Problem 3 is more complex than Problem 2. Also Problem 2 is more complex than Problem 1. From these results we can conclude the following:

1. Increasing the Elitism Size increases the probability to reach a solution till ESmax, and then it drops. This means that it is good to keep some of the best candidates found aside, but after some point and when the size of these kept candidates is getting bigger, i.e. less genetic operations are done, the diversity of the search space drops, hence the probability to find a solution decreases.
2. If we look at the optimum points(ESmax for each problem), we can find that ESmax for the Problem 1 is at 70%. ESmax for Problem 2 and Problem 3 is at 50%. This means that as the problem gets more complex, ESmax gets smaller. This means that as a problem grows and becomes more complex, more diversity is needed in the search space.

6.2 Tournament Size

Figure.12 shows the Tournament Size against probability to reach a solution. As a general behavior, and for the three problems in hand, increasing the Tournament Size increases the probability to reach a solution till some point, then

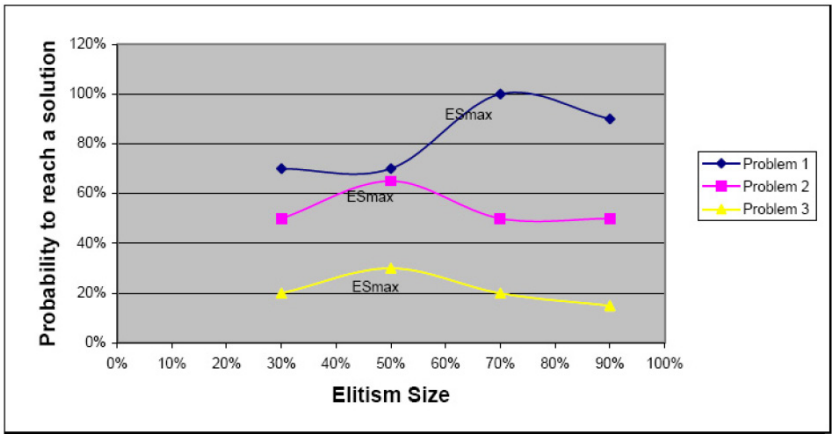

Fig. 11. Elitism Size against probability to reach a solution (No of solutions out of 20 different runs)

it remains almost constant. It is clear that the larger the Tournament Size is, the more likely we are to select a highly fit individual from the population, and hence we reach a solution faster.

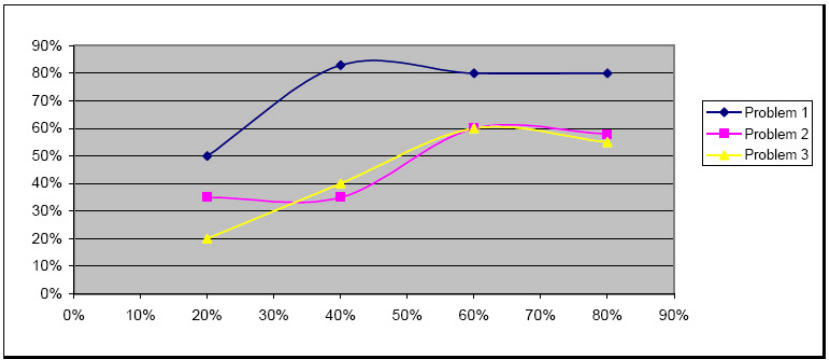

Fig. 12. Tournament Size against probability to reach a solution (No of solutions out of 20 different runs)

7 Comparison between PGen and Spock

As described previously, PGen is part of the Kirk model-based executive for mobile autonomous systems. PGen acts as the generative planner inside Kirk; its main role is to take a goal plan and form a solution plan by combining the goal

plan with a set of activities from the activity library and search for a consistent and complete solution plan using Genetic Algorithms. Among all work done in this area, we see that the most similar work done was Spock[7]. It is therefore worthy to compare our results with Spock's. Table.1 shows Spock results.

Table 1. Spock Results

Problem	*Events in Solution*	*Episodes in Solution*	*Time to Solve*
1	6	5	0.04s
2	10	13	0.14s
3	9	11	0.14s
4	11	13	0.11s
5	16	32	0.67s
6	20	44	2.35s
7	16	30	15.21s

Table 2. PGen Results

Problem	*Events in Solution*	*Episodes in Solution*	*Time to Solve*
1	8	10	0.07s
2	10	12	0.07s
3	9	11	0.13s
4	11	13	0.13s
5	14	15	0.27s
6	25	31	0.69s
7	27	34	2.21s

In general, PGen's performance[1] was not less than Spock. Moreover, more complicated problems were tested on PGen. To conclude, we find our planner presented in this paper is better than Spock for the following reasons:

1. PGen was run on 66 different test problems while Spock was run on just 7 test problems. So, PGen has been tested more intensively.
2. Full and complete performance analysis was presented for PGen while not any was presented for Spock
3. The activity library used in PGen's test problems consists of 43 activities while the one used for Spock consists of just 2 activities. This shows how solid the test phase prepared for PGen was, and how simple the test phase prepared for Spock was. Hence, because PGen was exposed to more complex

missions in testing, this makes it more reliable than Spock. We cannot judge at the moment how Spock will react when it is exposed to these complicated missions that PGen was exposed to.
4. Spock's evaluation function is not complete; it does not include a heuristic cost estimate. PGen's fitness function is complete and more robust.
5. Wide range of results was given for PGen. 6 different results were presented while Spock results were very poor.

8 Conclusion and Future Work

PGen main goal is to produce generative plans for complex processes. Goal plans and activity models are encoded using RMPL, while plan operators and plan candidates are represented by TPN. PGen uses Genetic Algorithms as a novel approach for TPN-based planning and it showed successful performance. Results are better than Spock; the most similar work done in this area. It is planned to run PGen on larger problems in order to demonstrate its applicability to more complex autonomous vehicle control scenarios. PGen is part of Kirk model-based executive for mobile autonomous systems, however for the time being it was tested separately without being integrated with Kirk. Its input was a TPN control program and its output was a complete and consistent TPN plan. Integration with the rest of the Kirk model-based executive will be completed in the near future.

References

1. Nermeen M.Ismail, Magda B.Fayek, Ashraf A.Wahab , Nevin M. Darwish *Planning Complex Processes for Autonomous Vehicles by Means of Genetic Algorithms. M.Eng.Thesis* (2008)
2. John R. Koza *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (1992)
3. Jinghui Zhong, Xiaomin Hu, Min Gu and Jun Zhang *Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms* (2005)
4. Gregory Horvath, Michel Ingham, Seung Chung, Oliver Martin, Brian Williams *Practical Application of Model-Based Programming and State-Based Architecture to Space Missions* (2006)
5. Seung H. Chung *Model-based Programming for Cooperative Vehicles: Generative Activity Planner* (2004)
6. Paul E. Black *Dictionary of Algorithms and Data Structures* (2004)
7. Jonathan Kennell *Generative Temporal Planning with Complex Processes. M.Eng.Thesis* (2003)
8. Brian C. Williams , Michel D. Ingham , Seung H. Chung and Paul H. Elliott *Model-based Programming of Intelligent Embedded Systems* (2003)
9. Philip K.Kim , Brain C. Williams and Mark Abramson *Executing Reactive Model-based Programs through Graph-based Temporal Planning* (2001)
10. Andreas F. Wehowsky , John Stedl and Brian C. Williams *Planning for Communication Between Cooperative Mars Rovers* (2001)

11. B. C. Williams and V. Gupta *Unifying Model-based and Reactive Programming in a Model-based Executive* (1999)
12. R. Dechter, I. Meiri, and J. Pearl *Artificial Intelligence* (1991) 49:61-95