C-FOREST: Parallel Shortest-Path Planning with Super Linear Speedup

Michael Otte
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139-4307

Nikolaus Correll
University of Colorado
430 UCB
Boulder, CO 80309-0430

Abstract

In (Otte and Correll 2013) we present C-FOREST, a parallelization framework for single-query sampling-based shortest-path planning algorithms. C-FOREST has been observed to have super linear speedup on many problems, e.g., paths of quality L_{target} are found 350X faster by 64 CPUs working in parallel than by 1 CPU. In (Otte and Correll 2013) C-FOREST is tested in conjunction with the RRT* algorithm. In the current work we perform additional experiments that show C-FOREST provides similar advantages when used conjunction with the SPRT algorithm. This reinforces our original claim that C-FOREST is generally applicable to a wide range of sampling based motion planning algorithms.

In (Otte and Correll 2013) we present a parallelization algorithm for single-query[1] shortest-path planning called *Coupled Forest Of Random Engrafting Search Trees*, or C-FOREST for short. C-FOREST is designed to be used with a distributed architecture in which T CPUs communicate via message passing. Quoting (Otte and Correll 2013):

> "Each CPU builds a *different* search tree between the *same* start and goal states, similar to OR-parallelization (Caselli and Reggiani 1999). Although most growth is random and independent, message passing enables new exploration and pruning *of all trees* to be a function of the current best solution known to any tree in the forest, unlike OR-parallelization, in which each tree grows completely independently. Nodes from solution branches are also exchanged so they can be engrafted onto and improved by the other trees. The latter allows good solutions to be improved by all trees in the forest, and provides all trees increased visibility of the configuration space."

Being a *parallelization framework*, C-FOREST is more akin to OR-parallelization than to RRT* or PRM. Indeed, it is not *itself* a path-planning algorithm but must be used on top of a predefined sampling based motion planning algorithm and in a space such that (1) the planning algorithm has almost sure (or better) convergence to the optimal solution, and (2) the triangle inequality holds in the space.

The *Path planning* problem is defined as finding an action sequence that causes a system to transition from an initial state to a goal state while avoiding obstacles. Formally, find $\mathbf{P} = \{\Delta x_1, ..., \Delta x_j\}$, where $x_{\text{goal}} = x_{\text{start}} + \sum_{i=1}^{j} \Delta x_i$, s.t. $(x_i, x_{i+1}) \cap X_{\text{obstacle}} = \emptyset$, where $x_{\text{k}} = x_{\text{start}} + \sum_{i=1}^{k} \Delta x_i$ and $x_{\text{start}} \equiv x_0$, and $X_{\text{obstacle}} \subset X$ is the subset of space X containing obstacles, and (x_i, x_{i+1}) is the trajectory through X from x_i to x_{i+1} that is followed by the robot's state as a result of action Δx_i. In *shortest-path planning* we desire to find the optimal path through X with respect to a distance function $\|\cdot\|$. Formally, find $\mathbf{P}^* = \text{argmin}_{\mathbf{P}} \sum_{i=0}^{j} \|x_{i+1} - x_i\|$.

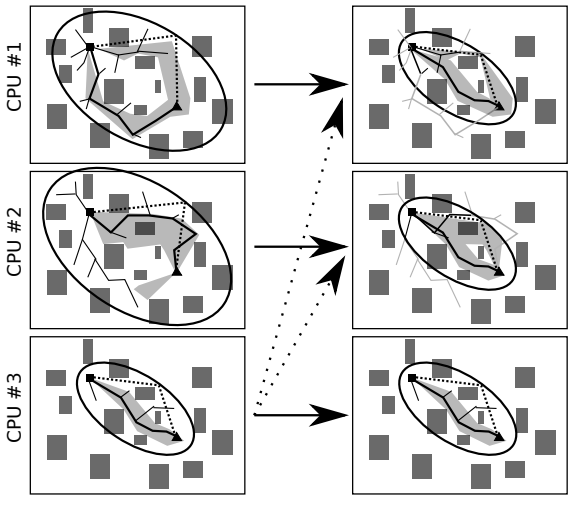

Figure 1: C-FOREST. CPUs build i.i.d. trees. Obstacles are dark gray, best-paths are bold and have the same length as the dashed lines that define the ellipsoids. #3 finds a better path it shares it (arrows) with #1 and #2. All CPUs prune edges (light gray) and avoid sampling outside the ellipsoid. The chances each CPU finds a better path increase (volume of light gray regions vs. ellipsoids).

[1] *Single-query* planners assume a new configuration space is encountered each time they are called (Sucan and Kavraki 2010), in contrast *multi-query* planners assume the same configuration space is used each time they are called (Overmars and Svestka 1995; Koga and Latombe 1994; Sanchez and Latombe 2002a; 2002b; Clark, Rock, and Latombe 2003).

Related Work

The path planning problem is well suited for algorithms that use parallelization. Probabilistic Road-Map (PRM) has been shown to be 'embarrassingly' parallel on memory shared architectures (Amato and Dale 1999). A similar idea is presented in (Ichnowski and Alterovitz 2012) for RRT and RRT*. Related algorithms have also been implemented on a GPU (Bialkowski, Karaman, and Frazzoli 2011), and with a shared memory architecture (Sucan and Kavraki 2009). OR-parallelization over a message passing architecture is used for feasible path-planning in (Caselli and Reggiani 1999; Challou, Gini, and Kumar 1993). The *Sampling-based roadmap of trees* (*SRT*) feasible-path planning algorithm has also been implemented on a message passing (Plaku et al. 2005). Multiple trees have been used for path planning with non-distributed architectures in (Li and Shie 2002; Gayle, Klingler, and Xavier 2007; Zucker, Kuffner, and Branicky 2007; Ferguson and Stentz 2006; Wedge and Branicky 2008). A detailed discussion of the differences between C-FOREST and all of these works appears in (Otte and Correll 2013). The latter extends our earlier work (Otte and Correll 2010a; 2010b) to shortest-path planning over message-passing architectures, in general. The current extended abstract contains the results of experiments that have not previously appeared outside the first author's PhD dissertation (Otte 2011).

Super Linear Speedup vs. Stopping Criterion

Let the wall time required to solve a particular problem with 1 or T CPUs be denoted w_1 and w_T, respectively. *Speedup* is defined $S = w_1/w_T$ and *efficiency* is defined $\eta = S/T$. Speedup measures the relative benefit of using T CPUs in parallel (i.e., with respect to elapsed time until the solution is found), while efficiency is inversely proportional to the amount of total computation power used to solve a problem.

The following discussion on stopping criteria appears in (Otte and Correll 2013):

> "C-FOREST will normally be used with an 'any-time' stopping criterion—it will search for better solutions as long as possible, given constraints imposed by safety, time and/or energy expenses, etc. However, it can alternatively use a cost-based stopping criterion—running until the first solution better than a predefined target cost is found, i.e., until $\|\mathbf{P}_{\mathrm{bst}}\| < L_{\mathrm{target}}$, where $\mathbf{P}_{\mathrm{bst}}$ is the best solution so far.
>
> By definition, S and η cannot be calculated when an any-time stopping criterion is used (for any algorithm)—since the any-time stopping criterion sets w_1 and w_T based on external factors.

All discussions about speedup and efficiency in this extended abstract refer to the L_{target} stopping criterion. That said, the qualitative results obtained with the L_{target} transfer to the any-time stopping criterion because they show how much more (or less) planning time we expect would be required to get the same result using a different T.

Experiments presented in (Otte and Correll 2013) show that C-FOREST can have super linear speedup ($S > T$, and $\eta > 1$) when the L_{target} stopping criterion is used with RRT*. Indeed, $\eta > 9$ is observed. In the current extended abstract of our broader work, we reproduce similar results using the SPRT algorithm.

```
in parallel /* on each CPU */
    while not stopping criteria met do
        x = random sample configuration space
        if x can safely connect to T then
            connect x to T
        if ||P_bst|| < L_old then
            broadcast P_bst
        if received message with P : ||P|| < ||P_bst|| then
            connect all x ∈ P \ T to T
        L_old = ||P_bst||
return P_bst
```

Figure 2: The C-FOREST Algorithm. The construction of $\mathcal{T}$ depends on the planning algorithm being used (e.g, RRT*). We assume asymptotically optimal convergence of $\|\mathbf{P}_{\mathrm{bst}}\|$.

C-FOREST Algorithm

An outline of the C-FOREST algorithm appears in Figure 2. A detailed description appears in (Otte and Correll 2013).

Each CPU builds an *independent and unique* random search tree $\mathcal{T}$ between the *same* start and goal states. Whenever a shorter path $\mathbf{P}_{\mathrm{bst}}$ is found (i.e., on a particular CPU), then it is sent to every other tree via message passing so that the nodes $x \in \mathbf{P}_{\mathrm{bst}}$ of its best path can be inserted into the trees on all other CPUs.

As originally explained in (Otte and Correll 2013), the latter exchange is beneficial in three ways: "(1) All trees expand into regions of the configuration space that are known to be beneficial by at least one tree. (2) All trees can prune themselves of globally outdated nodes and (3) focus their search by avoiding regions of the configuration space that cannot produce globally better solutions." For example, assuming the existence of a distance function $\|\cdot\|$, by pruning/ignoring all points $x \in X$ such that $\|x_{\mathrm{goal}} - x\| + \|x - x_{\mathrm{start}}\| < \|\mathbf{P}_{\mathrm{bst}}\|$.

C-FOREST's runtime is a combination of the inherited runtime, per node, of the underlying random tree algorithm and the time associated with sending/receiving messages. The former is $\mathcal{O}(c\log(n))$ for RRT* and $\mathcal{O}(n)$ for SPRT (where n is the number of nodes in $\mathcal{T}$, while c is a predefined constant used in RRT*). Let $g(n)$ be the time required to prune nodes from the tree based on $\|\mathbf{P}_{\mathrm{bst}}\|$, and let $f(i)$ be the time required to insert the i-th node of $\mathbf{P}_{\mathrm{bst}}$ into the tree. For RRT* $f(i) = \mathcal{O}(c\log(i))$ and for SPRT $f(i) = \mathcal{O}(i)$. Sending a message requires time $\mathcal{O}(\ell)$, while receiving a message requires time $\mathcal{O}(\ell + g(n) + \sum_{i=1}^{\ell} f(i))$.

Experiments

In (Otte and Correll 2013) we perform five sets of experiments with RRT* involving four different planning problems. In this extended abstract we present two additional ex-

periments in which C-FOREST is used with the SPRT algorithm: (1) A 7-DOF manipulator arm must plan around an obstacle and distance is defined as the total distance traveled by the end-effector of the arm. (2) A centralized[2] multi-robot team of four robots must plan in an office environment, and distance is defined as the euclidean distance through the combined configuration space of all robots.

The main difference between SPRT, which appears in Appendix B of (Otte 2011), and RRT* is that the former attempts direct connection of a new sample to every node already in the tree and propagates any resulting cost-to goal changes out to the search-tree's leaves (in order to maintain a consistent graph), while the latter only attempts connection to $O(\log(n))$ nearest neighbors in expectation and propagates cost-to-goal changes only to those $O(\log(n))$ neighbors (yielding an inconsistent graph). Thus, SPRT has a per-iteration runtime of $O(n)$ while RRT* has a per-iteration runtime of $O(\log n)$. On the other hand, cost-to-goal information updates much more quickly in SPRT than in RRT* due to graph consistency vs. inconsistency.

C-FOREST with SPRT is compared to OR-parallelization[3] with SPRT in order to compare the advantages of sharing data *during planning* vs. the purely statistical benefits of drawing multiples samples from the space of all random paths.

Our parallel architecture contains 64 single CPU computers (each with 1.2 GHz Xeon processors and Ubuntu operating system). Communication is over a network, and each CPU builds a single SPRT tree. We perform 32 repeated trials per datapoint for each combination of planning problem, parallelization method {C-FOREST, OR-parallelization}, forest sizes $T = \{1, 2, 4, 8, 16, 32, 64\}$, and stopping criterion L_{target} of varying difficulty. Note that when $T = 1$ the resulting algorithm is equivalent to using SPRT on a single tree without parallelization. Figures 3 and 4 show results.

Discussion of Results

The results observed with SPRT are very similar to those previously observed with RRT* in (Otte and Correll 2013). While we point the reader to (Otte and Correll 2013) for a detailed discussion, a summary appears below. Note that sentences within quotation marks are taken directly from (Otte and Correll 2013) and apply to experiments with RRT* as well as to experiments with SPRT.

C-FOREST outperforms OR-parallelization on shortest-path planning problems, in general, and has better performance as L_{target} decreases. "This reflects the fact that, before the initial solution is found, there is no difference between C-FOREST and OR-parallelization." "Intuitively, exchanging paths is beneficial because bottlenecks and other

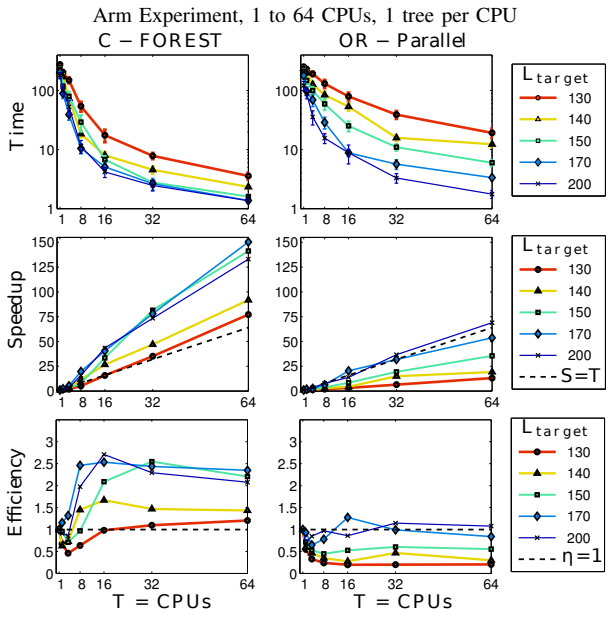

Figure 3: C-FOREST with SPRT vs. OR with SPRT for a 7 DOF arm. Color is target path length (L_{target}). Dashed lines show linear speedup ($\eta = 1$ and $S = T$) with respect to CPUs. Time (mean, standard error), speedup, and efficiency using 1-64 CPUs, 1 tree per CPU (32 runs per data-point).

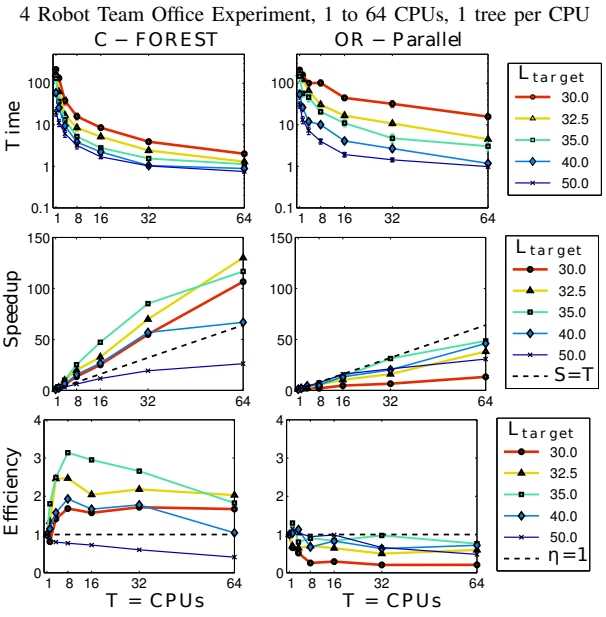

Figure 4: C-FOREST with SPRT vs. OR with SPRT, 4 robot team in an office. 32 runs per data-point. Same key as Fig 3.

obstructions can make exploring the space difficult, and finding a close-to-optimal solution unlikely." Coupled sampling allows all trees to "focus effort on exploring only regions of the configuration that can possibly lead to better solutions.

[2]All robot are viewed as individual pieces of a single larger robot in centralized multi-robot planning. The configuration space $\mathbb{C}$ of the single "meta-robot" is defined as the Cartesian product over the set of configuration spaces $\{\mathbb{C}_1, \ldots \mathbb{C}_R\}$ from each individual robot $1, \ldots, R$. Formally, $\mathbb{C} = \mathbb{C}_1 \times \mathbb{C}_1 \times \ldots \times \mathbb{C}_{R-1} \times \mathbb{C}_R$.

[3]OR-parallelization grows T trees in parallel that *do not communicate* during search

Coupled pruning reduces the amount of work required to insert new nodes into each tree."

While larger forests always find a target solution more *quickly*, the best *efficiencies* are observed when $T < 64$. An explicit method of optimizing T is beyond the scope of (Otte and Correll 2013) as well as the current work; however, performance appears to be stable vs. T; changing T by 30% has a relatively small effect on solution quality vs. time. In practice, we believe that a reasonable approximation of the optimal value of T can be found by tuning the parameter problems that have similar space topology and obstacle size/density.

C-FOREST's ability to achieve super linear speedup does not transfer to *feasible* path planning. In (Otte and Correll 2013) we evaluate C-FOREST on the Alpha-1.5 benchmark, a classic *feasible* planning problem, and find that speedups are nearly always sub-linear.

Conclusions

In this extended abstract we show that C-FOREST enables significant speedup when used with the SPRT algorithm. This compliments our previous results in (Otte and Correll 2013) where we showed that C-FOREST provides significant speedup with the RRT* algorithm. RRT* and SPRT have significantly different mechanics of convergence to the optimal solution (see Experiments Section for more information). Therefore, the current results support our claim that C-FOREST is widely applicable to many different sampling based motion planning algorithms. This is the first time that results regarding SPRT have appeared outside the first author's PhD Thesis.

References

Amato, N. M., and Dale, L. K. 1999. Probabilistic roadmap methods are embarrassingly parallel. In *Proc. IEEE International Conference on Robotics and Automation*, 688–694.

Bialkowski; Karaman; and Frazzoli. 2011. Massively parallelizing the RRT and the RRT*. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Caselli, S., and Reggiani, M. 1999. Randomized motion planning on parallel and distributed architectures. In *Euromicro Workshop on Parallel and Distributed Processing*, 297–304.

Challou, D. J.; Gini, M.; and Kumar, V. 1993. Parallel search algorithms for robot motion planning. In *IEEE International Conference on Robotics and Automation*, 46–51.

Clark, C. M.; Rock, S. M.; and Latombe, J.-C. 2003. Dynamic networks for motion planning in multi-robot space systems. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 3621–3631.

Ferguson, D., and Stentz, A. 2006. Anytime RRTs. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5369–5375.

Gayle, R.; Klingler, K. R.; and Xavier, P. G. 2007. Lazy reconfiguration forest (LRF) - an approach for motion planning with multiple tasks in dynamic environments. 1316–1323.

Ichnowski, J., and Alterovitz, R. 2012. Parallel sampling-based motion planning with superlinear speedup. In *In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Koga, Y., and Latombe, J.-C. 1994. On multi-arm manipulation planning. In *Proc. IEEE International Conference on Robotics and Automation*, volume 2, 945–952.

Li, T.-Y., and Shie, Y.-C. 2002. An incremental learning approach to motion planning with roadmap management. In *IEEE International Conference on Robotics and Automation*, 3411–3416.

Otte, M., and Correll, N. 2010a. Any-com multi-robot path-planning: Maximizing collaboration for variable bandwidth. In *Proc. International Symposium on Distributed Autonomous Robotics Systems*.

Otte, M., and Correll, N. 2010b. Any-com multi-robot path-planning with dynamic teams: Multi-robot coordination under communication constraints. In *Proc. International Symposium on Experimental Robotics*.

Otte, M., and Correll, N. 2013. C-FOREST: Parallel shortest-path planning with super linear speedup. volume 29, 798–806.

Otte, M. 2011. *Any-Com Multi-Robot Path Planning*. Ph.D. Dissertation, University of Colorado at Boulder.

Overmars, M., and Svestka, P. 1995. A probabilistic learning approach to motion planning. In *Algorithmic Foundations of Robotics (WAFR)*, 19–37.

Plaku, E.; Bekris, K. E.; Chen, B. Y.; Ladd, A. M.; and Kavraki, L. E. 2005. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics* 21:587–608.

Sanchez, G., and Latombe, J.-C. 2002a. On delaying collision checking in prm planning: Application to multi-robot coordination. *The international Journal of Robotics Research* 21:5–26.

Sanchez, G., and Latombe, J.-C. 2002b. Using a prm planner to compare centralized and decoupled planning for multi robot systems. In *Proc. IEEE International Conference on Robotics and Automation*, volume 2, 2112–2119.

Sucan, I. A., and Kavraki, L. E. 2009. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII (Proceedings of Workshop on the Algorithmic Foundations of Robotics), STAR*, 449–464.

Sucan, I. A., and Kavraki, L. E. 2010. On the implementation of single-query sampling-based motion planners. In *IEEE International Conference on Robotics and Automation*, 2005–2011.

Wedge, N. A., and Branicky, M. S. 2008. On heavy-tailed runtimes and restarts in rapidly-exploring random trees. In *AAAI Conference on Artificial Intelligence*.

Zucker, M.; Kuffner, J. J.; and Branicky, M. S. 2007. Multipartite RRTs for rapid replanning in dynamic environments. In *International Conference on Robotics and Automation*, 1603–1609.